CDRL No. 0002AC-4

Software Design Specification for the Manufacturing Optimization (MO) System

Linda J. Lapointe
Thomas J. Laliberty
Robert V.E. Bryant

Raytheon Company

1992

DARPA
Defense Advanced Research
Projects Agency

CDRL No. 0002AC-4

Software Design Specification for the Manufacturing Optimization (MO) System

Prepared by
Linda J. Lapointe
Thomas J. Laliberty
Robert V.E. Bryant

Raytheon Company
Missile Systems Laboratories
Tewksbury, MA 01876

December 1992

ARPA Order No. 8363/02
Contract MDA972-92-C-0020

Prepared for

DARPA

Defense Advanced Research Projects Agency

Contracts Management Office
Arlington, VA 22203-1714

Contents

Figures and Tables

1. Scope

1.1 Identification

This is the Software Design Specification for the Manufacturing Optimization (MO) System. The development activities are being performed under Defense Advanced Research Projects Agency (DARPA) funding, contract number MDA972-92-C-0020, by the MO Development Team. The Development Team is comprised of personnel from Computer Aided Engineering Operations (CAEO) of the Raytheon Missile Systems Laboratories (MSL) with participation from the MSL Mechanical Engineering Laboratory (MEL) and the Missile Systems Division (MSD) West Andover Manufacturing facility.

1.2 System Overview

DICE has developed a concurrent engineering model that replicates the human tiger team concept. The basic tenet of the human tiger team is to have the various specialists contributing to the project co-located. In today's environment of complex product designs and geographically dispersed specialists, DICE envisioned a "virtual tiger team" working on a "unified product model" accessible by computer networks. Such an environment must enable specialists from each functional area to work on the design concurrently and share development ideas.

Raytheon has proposed a conceptual refinement to the original DICE virtual tiger team. This refinement is a two level approach with a product virtual team having a global view supported by information supplied by lower level "specialized" process virtual teams. See Figure 1.2-1. This refinement is needed because of the growing complexity of our products and supporting development processes, which make it difficult for one individual to adequately comprehend all of the complexities required to establish a unified manufacturing position. The "virtual process team" concept would allow comprehensive representation from each specialized process area to contribute to the formulation of the final manufacturing recommendations.

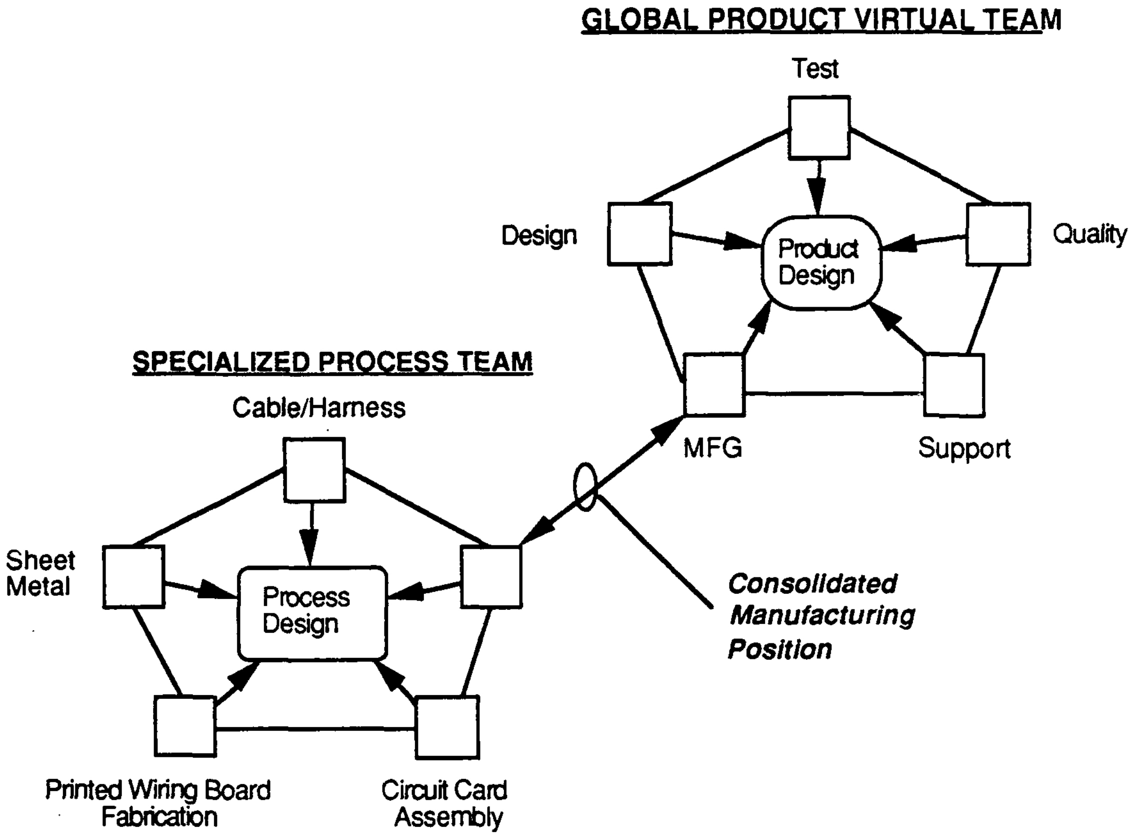


Figure 1.2-1 Two Level Team Concept

The purpose of the Manufacturing Optimization (MO) system is to enable all manufacturing specialists to participate in the product/process development activity concurrently. The system consists of a set of tools to model the manufacturing processes and centralize the various process tradeoffs. Recommendations can be compared and negotiated among the individual manufacturing participants. After the manufacturing team has reached a consolidated position, the results are passed back to the cross functional (top level) team for their negotiation.

1.3 Definition of Key Terms

Communications Manager (CM) - a collection of modules developed as part of the DICE program which facilitate distributed computing in a heterogeneous network.

Consolidated Manufacturing Position - recommendations from the manufacturing process team. The recommendations are in the form of product design changes. The changes will optimize the manufacturing process of the product for cost and yield.

Dependency Graph - a directed acyclic graph defined such that each node in the graph has a set of parent nodes that it is dependent on. In MO, the nodes in the graph represent individual processes. Each node in the graph contains an AND/OR flag. If the flag is set to AND then the node is dependent on all of its parents. If the flag is set to OR then the node is dependent on any one of its parents. For a particular process to be included in the overall manufacturing process for a particular product, the nodes that it is dependent on must be satisfied.

Manufacturing Advisor - a MO core module which provides the user with various methods to view the results produced by the Manufacturing Analyzer. Results can be viewed via graphing functionality or through textual reports.

Manufacturing Analyzer - a MO core module which provides the following three services: 1. Selection of individual processes from the process model which are used to manufacture a particular product; 2. Analysis of the processes selected and the operations attached to each process to estimate scrap and rework rates; 3. Analysis of the resources needed to perform the operations attached to the selected processes for cost.

Operation - a unit of work performed on the part. Associated with each operation are scrap rates, rework rates, and required resources.

Project Coordination Board (PCB) - a DICE tool supported by CERC that provides support for the coordination of the product development activities in a cooperative environment. It provides for common visibility into the design task structure, task assignment capabilities, and change notification capabilities.

Process - an organized group of manufacturing operations sharing characteristics. In MO, the process is modeled as a collection of operations that have associated scrap and rework rates and a list of required resources. Attached to each process is a set of selection rules and a set of parent processes which the process is dependent on. For the process under consideration to be selected by the Manufacturing Analyzer, the parent processes that it is dependent on must have been selected and at least one of its selection rules must evaluate to true.

Process Model - the specification of the total manufacturing process required to produce the product. The process model consists of a directed acyclic dependency graph of individual manufacturing processes.

Process Modeler - a MO core module which provides the user with the ability to graphically model processes, operations, and resources.

Process Team - the lower level specialized team in the two-tiered team concept. The process team is responsible for providing a consolidated position in terms of their specialization. The users of the MO system would be part of the manufacturing process team and would be required to provide a consolidated manufacturing position to the global level product team.

Product Model - a set of STEP entities that define the features and attributes of the product. The Process Modeler provides a means of defining rules and equations in terms of the existence, count, or value of particular product model entity instances.

Product Team - the global level team in the two-tiered team concept. The global team is supported by all of the specialized process teams.

PWB - Printed Wiring Board.

RAPIDS - Raytheon Automated Placement and Interconnect Design System. Raytheon's conceptual design and analysis workstation for Printed Wiring Boards (PWB). RAPIDS supports component placement and placement density analysis, as well as a number of other analysis functions, including automatic component insertion checking. Interfaces between RAPIDS and the PWB analysis tools for the following criteria are also provided as part of the RAPIDS tool suite: Manufacturing, Post Layout Effects, Reliability, and Thermal.

Rapids to Step - a C++ application which utilizes the ROSE database and tools developed by STEP Tools Inc. The program reads the RAPIDS database using the RAPIDS Procedural Interface. A persistent STEP object of the appropriate class is generated for each RAPIDS record read. The object is then stored as a STEP entity in a physical STEP file.

Resource - any facility, labor, equipment, or consumable material used in the manufacturing process.

Rework Rate - the percentage of product parts which must be reworked due to an operation. Rework data is maintained in a list of rework entities. In each entity there is a rework rule and a corresponding rework rate. If the rework rule is satisfied, then the corresponding rework rate is computed. There is a list of resources associated with the rework which is used to calculate the cost of performing the rework operation.

Requirements Manager (RM) - Product Track Requirements Manager (CIMFLEX Teknowledge) is a software tool designed to manage product requirements and evaluate the compliance of product design data with requirements.

ROSE - Rensselaer Object System for Engineering is an object-oriented database management system developed at Rensselaer Polytechnic Institute. It has been developed to support engineering applications as part of the DICE program. ROSE is currently part of the STEP Programmer's Toolkit available from STEP Tools, Inc. ROSE is a database which supports concurrency using a data model that allows the differences between two design versions to be computed as a delta file. The MO data for the manufacturing processes and operations, as well as, the various analysis results will be stored and managed within ROSE.

STEP - STandard for the Exchange of Product model data is the International Standards Organization standard 10303. The objective of the standard is to provide a mechanism capable of representing product data throughout the life cycle of a product, independent of any particular system. STEP data is stored as instances of class entities.

Step to Rapids - a C++ application which utilizes ROSE and tools developed by STEP Tools Inc. The program reads a STEP file conforming to the EXPRESS schemas that model the PWB product data. The ROSE STEP filer is used to read the STEP file into instances of classes created by the express2c++ compiler. The class instance is then transformed into the appropriate RAPIDS data record and stored to the RAPIDS database.

Yield Rate - the percentage of product parts that must be scrapped due to an operation. Scrap data is maintained in a list of scrap entities. In each entity there is a scrap rule and a corresponding scrap rate. If the scrap rule is satisfied, then the corresponding scrap rate is computed.

1.4 Document Overview

The purpose of this report is to provide the software design for the Manufacturing Optimization (MO) System. It contains the preliminary and detailed object oriented design, the user interface design, and the schema specifications for MO.

The preliminary design discusses the capabilities and interfaces provided in the MO system. The detailed object oriented design describes the definition of the classes, objects, and methods which make up the MO system. The user interface design provides the look and feel of the system to the user, and the schema specification provides the details of the data behind the class and objects in the system.

2. Referenced Documents

1. BR-20558-1, 14 June 1991, DARPA Initiative In Concurrent Engineering (DICE) Manufacturing Optimization - Volume I - Technical.

2. CDRL No. 0002AC-1, March 1992, Operational Concept Document For The Manufacturing Optimization (MO) System, Contract No. MDA972-92-C-0020.

3. CDRL No. 0002AC-2, March 1992, Description of CE Technology For The Manufacturing Optimization (MO) System, Contract No. MDA972-92-C-0020.

4. CDRL No. 0002AC-3, May 1992, Functional Requirements and Measure of Performance For The Manufacturing Optimization (MO) System, Contract No. MDA972-92-C-0020.

5. Object-Oriented Analysis, Second Edition by Peter Coad/Edward Yourdon, Yourdon Press Computing Series, 1991.

6. Object-Oriented Design by Peter Coad/Edward Yourdon, Yourdon Press Computing Series, 1991.

7. Object Oriented Design with Applications by Grady Booch, The Benjamin/Cummings Publishing Company, Inc., 1991.

8. Product Data Representation and Exchange–Part 11: The EXPRESS Language Reference Manual, ISO DIS 10303-11, National Institute of Standards and Technology, 1992.

9. ProductTrack Requirement Manager User Guide and Reference, Release 1.02 for Sun SPARC and Oracle RDBMS, Cimflex Teknowledge Corporation, October 1992.

10. RAPIDS Database Data Dictionary, RAYCAD Document #1266021, Raytheon Company, November 22, 1991.

11. STEP Programmer's Toolkit Reference Manual, STEP Tools Inc., 1992.

12. STEP Programmer's Toolkit Tutorial Manual, STEP Tools Inc., 1992.

13. STEP Utilities Reference Manual, STEP Tools Inc., 1992.

14. User Manual for the Project Coordination Board (PCB) of DICE (DARPA Initiative in Concurrent Engineering, July 10, 1992.

3. Preliminary Design

3.1 MO Overview

The concept of the Manufacturing Optimization (MO) system is to facilitate a two tiered team approach to the product/process development cycle where the product design is analyzed by multiple manufacturing engineers and the product/process changes are traded concurrently in the product and process domains. The system will support Design for Manufacturing and Assembly (DFMA) with a set of tools to model the manufacturing processes and manage tradeoffs across multiple processes. The lower level "specialized" team will transfer their suggested design changes back to the top-level product team as the Manufacturing Team's consolidated position.

The external software packages which the MO system is comprised of are the ROSE DB, Requirements Manager, and the Project Coordination Board/Communications Manager. For demonstration purposes, an interface was developed between Raytheon Automated Placement and Interconnect Design System (RAPIDS) and the ROSE DB. Figure 3.1-1 illustrates the external interfaces to the MO system.

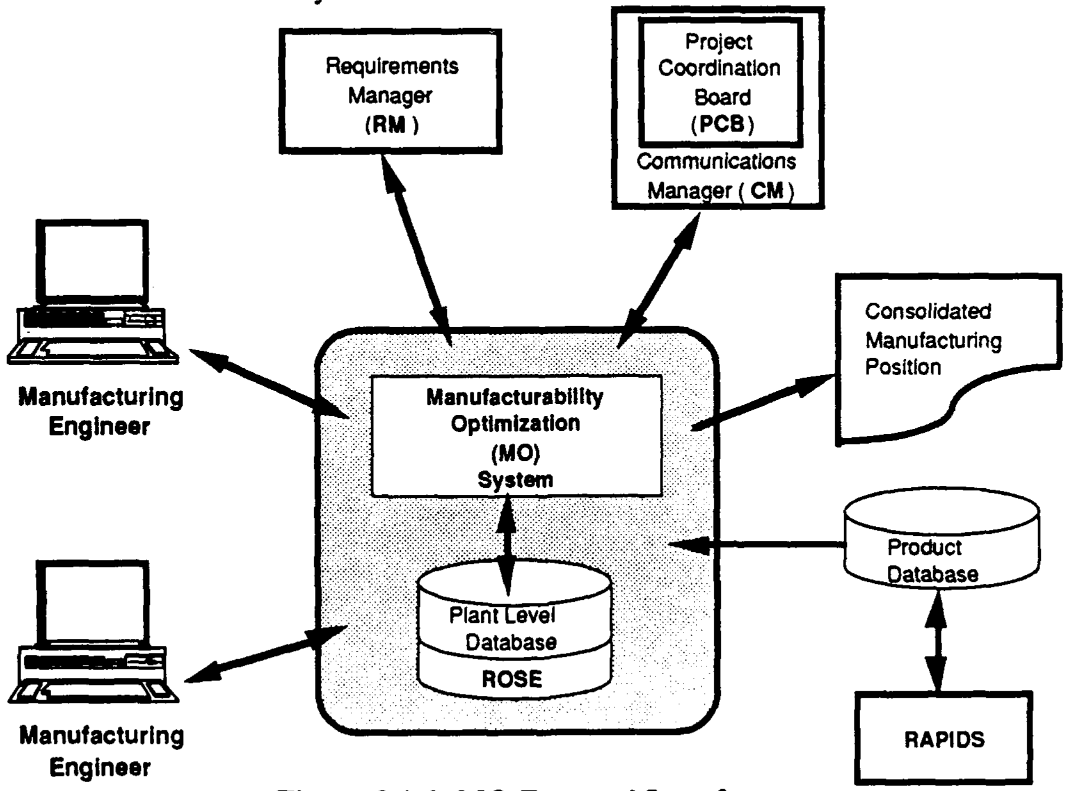

Requirements Manager (RM)
Project Coordination Board (PCB)
Communications Manager (CM)
Consolidated Manufacturing Position
Manufacturing Engineer
Manufacturability Optimization (MO) System
Plant Level Database
ROSE
Product Database
RAPIDS
Manufacturing Engineer

Figure 3.1-1 MO External Interfaces

ROSE is an object-oriented database management system that has been developed for engineering applications and enhanced to support the DICE program. ROSE is currently part of the STEP Programmer's Toolkit from STEP Tools, Inc. ROSE is a database which supports concurrency using a data model that allows the differences between two design versions to be computed as a delta file. The MO data for the manufacturing processes and operations, as well as, the various analysis results will be stored and managed within ROSE. The manufacturing process data consists of the process selection knowledge base, process and operation data, yield and rework data, and resource specifications.

The Requirements Manager (RM) is a software tool designed to manage product requirements and evaluate the compliance of product design data with requirements. The purpose of integrating the RM into the MO system is to provide the "top level" product development team insight into manufacturing requirements. It is common practice for a manufacturer to document manufacturability, or producibility guidelines which delineate standard manufacturing practices and acceptable design parameters. The purpose of these guidelines is to communicate the capabilities of the manufacturing process to the product design community to ensure that new product designs are specified within manufacturing capabilities. The guidelines delineate quantitative and qualitative producibility issues. The RM and the MO software will be tightly coupled through the RM's Application Programming Interface (API) to provide the user with a manufacturing guidelines analyzer capability.

The Project Coordination Board (PCB) provides support for the coordination of the product development activities in a cooperative environment. It provides common visibility and change notifications. The Communications Manager (CM) is a collection of modules that facilitates distributed computing in a heterogeneous network. The Communication and Directory Services provided in the CM module are required to utilize the PCB. The PCB/CM will be used in MO to support the communication of the product/process development activities. There will be no direct interface between the MO software modules and the PCB/CM applications. It will be used to manage the product task structure.

RAPIDS is Raytheon's conceptual design and analysis workstation for Printed Wiring Boards (PWB). RAPIDS supports component placement and placement density analysis, as well as a number of other analysis functions, including automatic mponent insertion checking and thermal analysis.

3.1.1 MO Architecture

MO is a X-Windows based tool. The application software will be written in C++, the user interface will be developed using OSF/Motif Widgets, and all data will be stored in STEP physical files.

The decision to use STEP physical files for the underlying data format for the MO system stems from the fact that STEP is the emerging international standard for data exchange between automation systems. Access to these STEP files will be provided through the STEP Programmer's Toolkit from STEP Tools Inc. The Toolkit provides a means of reading and writing STEP entity instances through a C++ class library.

The MO core system is composed of three software modules, Manufacturing Analyzer, Manufacturing Advisor, and Process Modeler. The Project Coordination Board (PCB) and Communications Manager (CM) from Concurrent Engineering Research Center (CERC), ProductTrack Requirements Manager (RM) from Cimflex Teknowledge, the ROSE database from STEP Tools Inc., and the two way interface to the Raytheon Automated Placement and Interconnect Design System (RAPIDS) complete the software suite which constitute the MO system. Figure 3.1-2 illustrates the MO System Architecture.

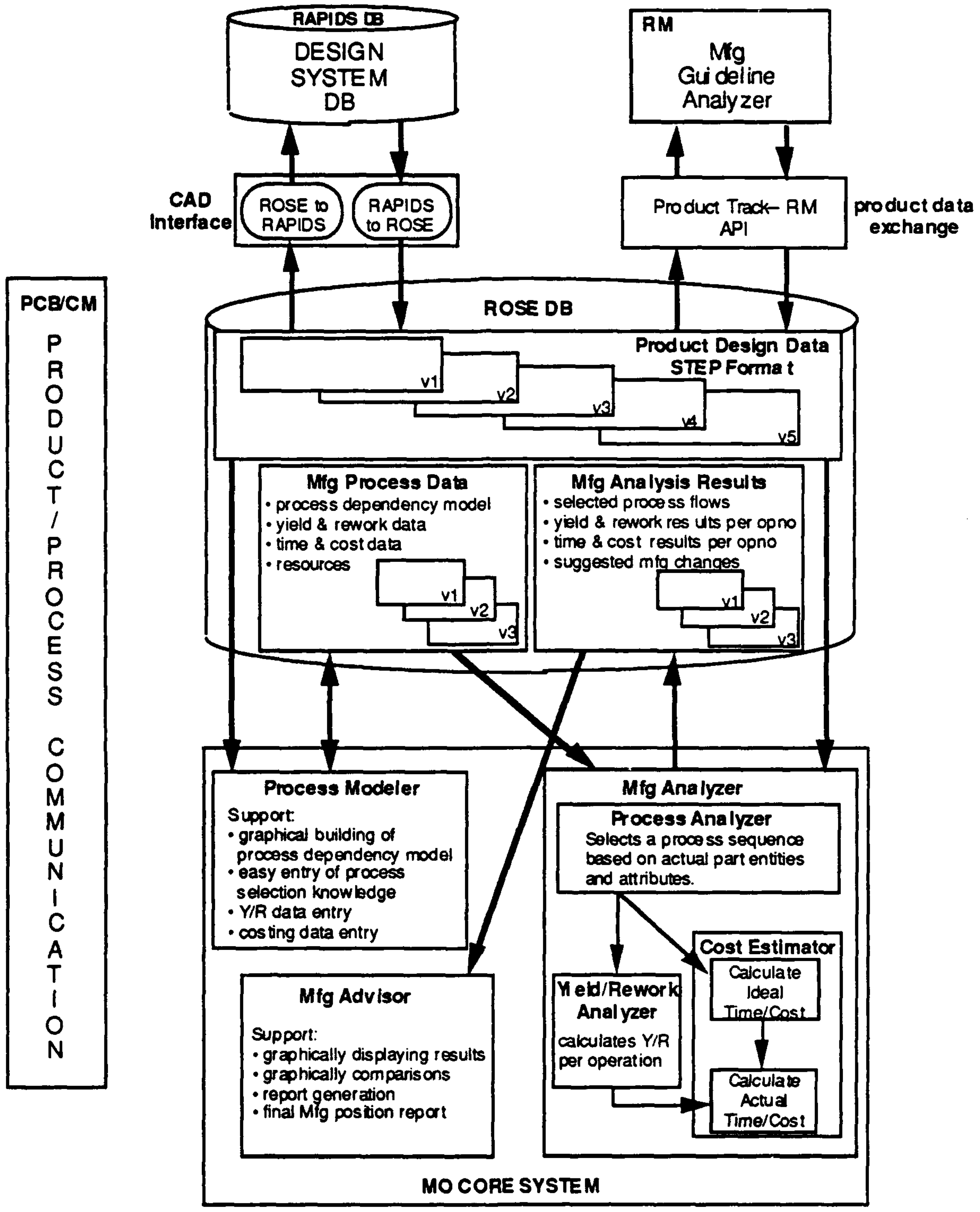

RAPIDS DB
DESIGN SYSTEM DB
RM
Mfg Guideline Analyzer
CAD Interface
ROSE to RAPIDS
RAPIDS to ROSE
Product Track- RM API
product data exchange
PCB/CM
PRODUCT/PROCESS COMMUNICATION
ROSE DB
Product Design Data STEP Format
v1
v2
v3
v4
v5
Mfg Process Data
• process dependency model
• yield & rework data
• time & cost data
• resources
Mfg Analysis Results
• selected process flows
• yield & rework results per opno
• time & cost results per opno
• suggested mfg changes
Process Modeler
Support:
• graphical building of process dependency model
• easy entry of process selection knowledge
• Y/R data entry
• costing data entry
Mfg Analyzer
Process Analyzer
Selects a process sequence based on actual part entities and attributes.
Cost Estimator
Calculate Ideal Time/Cost
Yield/Rework Analyzer
calculates Y/R per operation
Calculate Actual Time/Cost
Mfg Advisor
Support:
• graphically displaying results
• graphically comparisons
• report generation
• final Mfg position report
MO CORE SYSTEM

Figure 3.1-2 MO System Architecture

The Process Modeler provides the user with the ability to model processes required to manufacture a product. Each process is modeled as a set of operations, where an operation is a unit of work performed on the product part. Resources, yield rates, and rework rates are defined for each operation. The output of the Process Modeler is a directed acyclic dependency graph of individual manufacturing processes. Figure 3.1-3 depicts a block diagram of the Process Modeler.

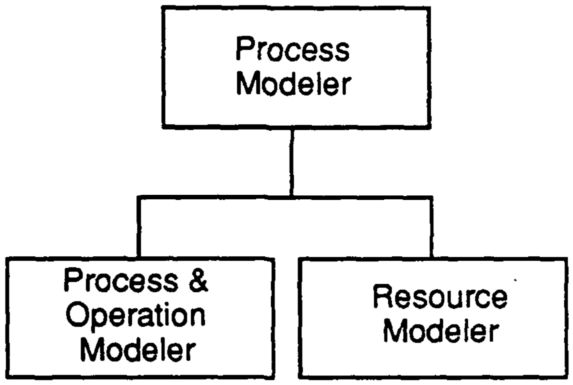


Figure 3.1-3 Process Modeler Block Diagram

Figure 3.1-4 illustrates a simplified subset of a process model. The example models the processes required to etch conductive material from a printed wiring board substrate.

The process modeler provides the ability to establish dependencies between processes. In the example, the "Etch Material" process is dependent on the "Etch FLEX" process OR the "Etch Substrate" process. Note that if the AND/OR flag of the "Etch Material" process were set to AND, then the "Etch Material" process would be dependent on both of its parents processes. Attached to the "Etch Material" process is a list of selection rules which provide the reason(s) why this process would be relevant, and a list of operations which must be performed to complete the etch material process. Defined with each operation are scrap rates, rework rates, setup time, run time, and a list of resources required to complete the operation.

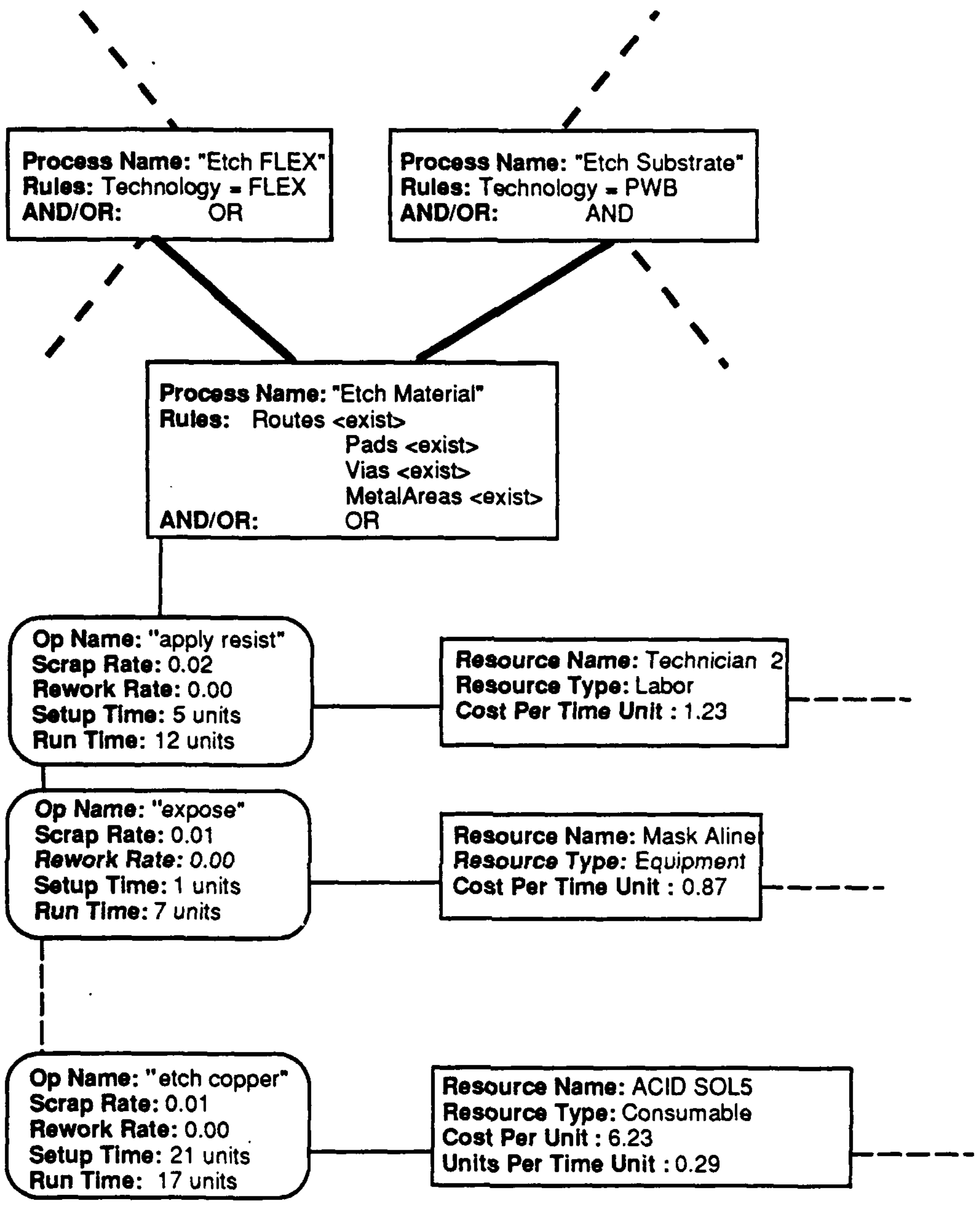


Figure 3.1-4 Example subset process model for etching process

The Manufacturing Analyzer provides the following three services: 1. Select the individual processes from the process model that are used to manufacture a particular product. 2. Analyze the processes selected and the operations attached to each process to estimate scrap and rework rates. 3. Analyze the resources needed to perform the operations attached to the selected processes for cost. The analyzer results are composed of design feature entities from the product design database (STEP file) along with the selected manufacturing processes from the user specified process model. Figure 3.1-5 depicts a functional block diagram of the Manufacturing Analyzer.

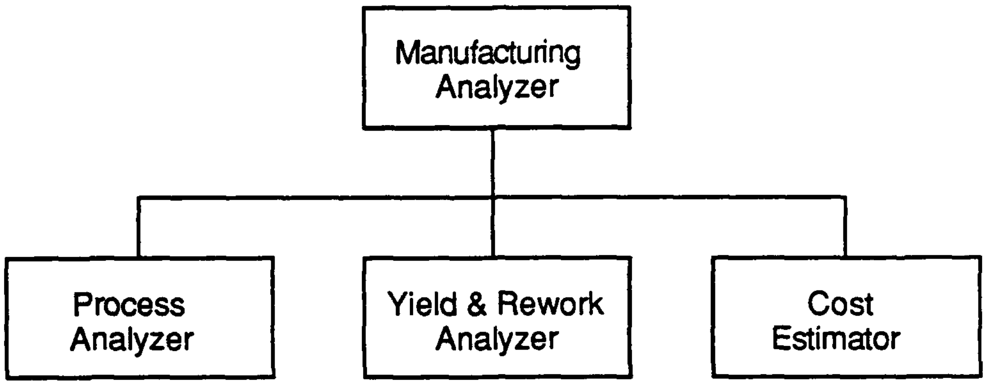


Figure 3.1-5 Manufacturing Analyzer Block Diagram

To illustrate all the Analyzer functionality (i.e. analysis of manufacturing process, yield, rework, and cost), lets utilize the example process model for the etch process defined in Figure 3.1-4. Lets walk through the steps that the process analyzer, yield/rework analyzer, and cost estimator will go through for determining selection of the "Etch Material" process. Consider that the product design data under analysis contains the following: an entity called Technology with a value of "PWB" and several Route, Pad, and Via entities.

To determine if the "Etch Material" process should be selected, the Process Analyzer will first determine if either of its parent nodes were previously selected as applicable processes. Since the product data contains a Technology entity set to PWB, the selection rules of the "Etch Substrate" process would have been satisfied; therefore, the process would have been previously selected. Since the AND/OR flag of the "Etch Material" process is set to OR only one of its parents must be selected before the Process Analyzer will evaluate its process selection rules. Since the product data set contains several routes, pads, and vias, the selection rules evaluate to true, thereby, satisfying all the process selection criteria. At that point, the process analyzer would add the "Etch Material" process to the resulting analysis flow along with the associated design feature entities from the product design.

Once the "Etch Material" process is selected, the Yield/Rework Analyzer will evaluate each operation attached to it to estimate scrap and rework rates associated with this particular product. Finally, the cost estimator will calculate the cost associated with the resources attached to each of the operations.

The Manufacturing Advisor provides the user with various methods to view the results produced from the analyzer. The results can be viewed graphically (i.e. line, bar, stacked bar and pie charts) or textually. The reporting capability allows the user to customize a detailed report which can be printed to the screen or to an ASCII file. MO allows the user to view one or more sets of analysis results at a time. By selecting multiple analysis runs to graphically display, the user can visually compare the analyses. Figure 3.1-6 shows a functional block diagram of the Manufacturing Advisor.

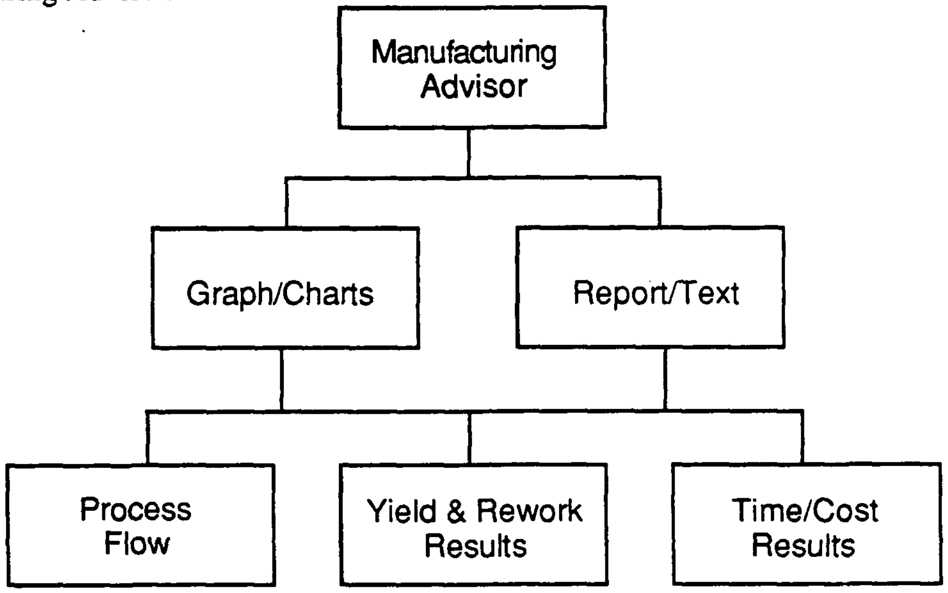


Figure 3.1-6 Manufacturing Advisor Block Diagram

The interaction between the user, manufacturing analyzer, process modeler, manufacturing advisor, RM, PCB/CM, RAPIDS, and ROSE DB is pictured in Figure 3.1-7.

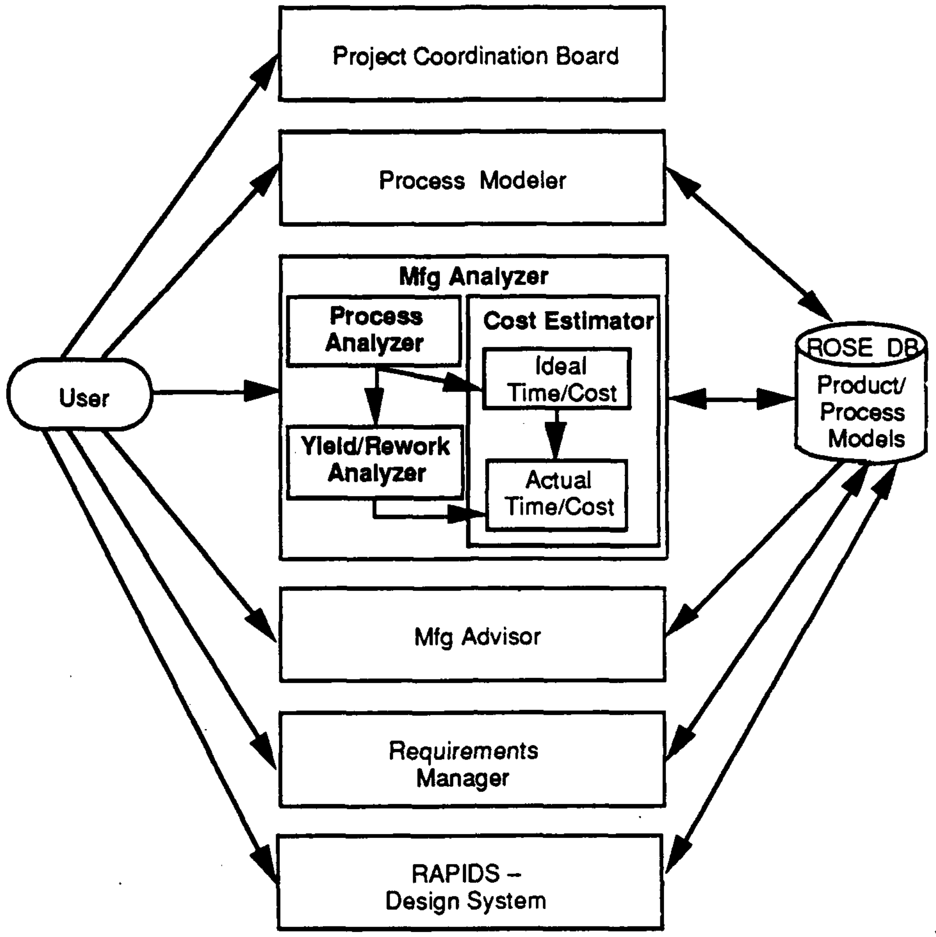


Figure 3.1-7 MO Data Flow

3.1.2 System States and Modes

At any time, MO will be in one of three modes: startup, active, or ready. Table 3.1-1 shows which buttons on the MO main window (refer to figure 4-1) are available for execution in each mode. Table 3.1-2 shows which buttons on the MO Advisor window (refer to figure 4.3-1) are available for execution in each mode. Table 3.1-3 shows which buttons on the MO Modeler window (refer to figure 4.4-1) are available for execution in each mode.

Table 3.1-1 Main Window State Execution

Mode	State	File	Analyzer	Advisor	Modeler
Startup	First enter MO	yes	no	no	yes
Active	Product/Process Models selected	yes	yes	no	yes
Ready	Analyzer has been run	yes	yes	yes	yes

Table 3.1-2 Advisor Window State Execution

Mode	State	Select	Process Graphs	Quality Graphs	Costing Graphs	Analysis Report	Exit to Main
All Modes	Entry	yes	yes	yes	yes	yes	yes

Table 3.1-3 Modeler Window State Execution

Mode	State	Models	Resources	Add	Delete	Exit to Main
Startup	First enter Modeler	yes	yes	no	no	yes
Active	Process Model selected	yes	yes	yes	no	yes
Ready	Process has been added	yes	yes	yes	yes	yes

The user will initially enter MO in the startup mode of the main window. The product and process models have not yet been selected; therefore, the analyzer and advisor are disabled. Once the user selects the applicable models, the analyzer becomes active. The user then executes the analyzer module. After the analyzer is complete, the advisor becomes active.

The user can then choose to view the analysis results by selecting the advisor button. The user will be put into the advisor window where all functionality is active.

At any time after starting up the MO system, the user can choose to enter the modeler startup window. Since no process model has been selected, add and delete are disabled. Once the user selects to create a new or modify an old process model, the add button becomes active. Once there are processes available for deletion, the delete button becomes active. At any time, the user can choose to return to the main MO window.

3.2 MO Design Description

3.2.1 External Interfaces

3.2.1.1 Project Coordination Board

The Project Coordination Board (PCB) is a system developed to provide support for the coordination of the product development activities in a cooperative environment. The PCB provides common visibility and change notification through the common workspace, planning and scheduling of activities through the task structure, monitoring progress of product development through the product structure (i.e. constraints), and computer support for team structure through messages. The Communications Manager (CM) is a collection of modules that facilitates distributed computing in a heterogeneous network. It promotes the notion of a virtual network of resources which the project team members can exploit without any prior knowledge of the underlying physical network. The Communication and Directory Services provided in the CM module are required to utilize the PCB.

MO introduces the concept of a two tiered virtual tiger team. The two tiered approach consists of a cross functional product team linked to teams within each of the functions, in this case a manufacturing process team. To implement this approach there must be communication among the members of each team, and between the product and process team. The PCB/CM is being used to support the following capabilities which are required for this type of communication:

- Product - to - Process Team Communication
  - Notification of design task completed.
  - Notification and issuance of database available for analysis.
  - Notification of alternative designs or trade-off decisions under consideration.
- Process - to - Product Team Communication
  - Notification and issuance of analysis results.
  - Notification and issuance of modified database with recommended changes.
  - Notification of changes to the process, guidelines, cost or yield models.

We are using the product task structure within the PCB/CM to model the product to process development team communication. Included in this task structure are major design steps, such as concept development, design capture, design verification, component placement, routing, transition to production, and several design reviews. The design reviews included representatives from design, test, reliability, manufacturing, and thermal. Figure 3.2-1 is a high level view which represents the design cycle steps which model a typical PWB product design cycle.

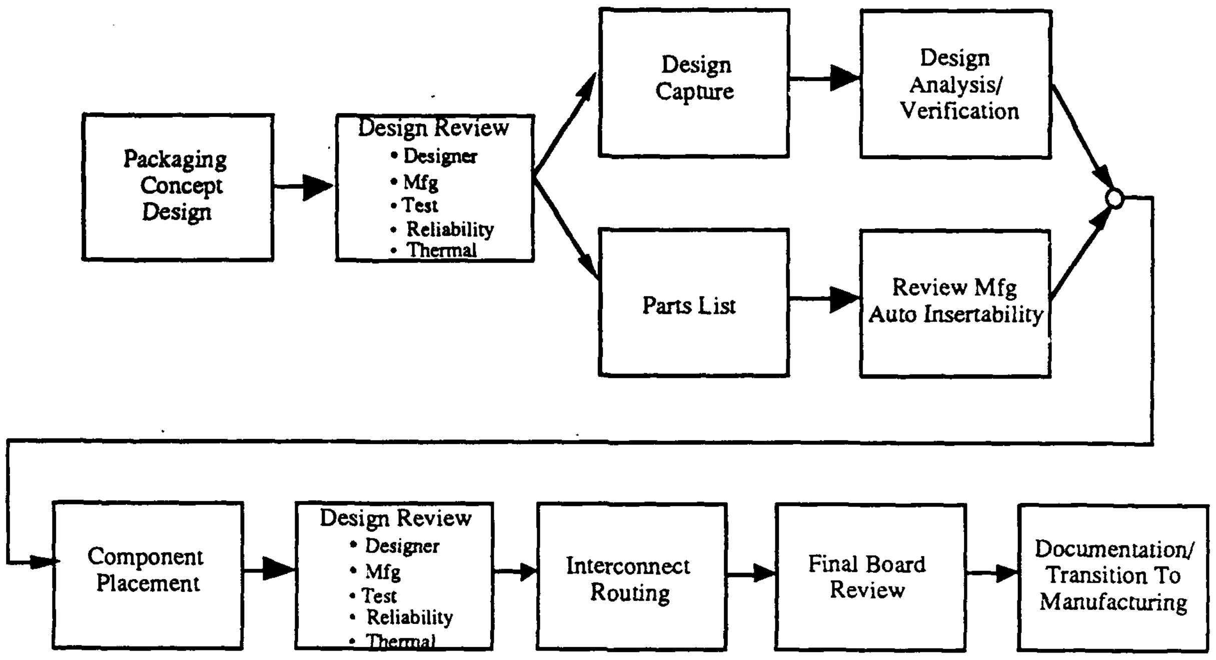


Figure 3.2-1 Sample PWB Design Cycle Flow

The Project Lead (user with special privileges) initializes the product task structure. The Project Lead can then view any task or work order that appears in the network, add a task to the existing network, acknowledge receiving a task, and indicate completion of a task. The other team members can acknowledge receiving a task and indicate completion of that task. The PCB automatically dispatches tasks as previous tasks are completed, as well as, the Project Lead can dispatch a task. Refer to Section 2 reference 14 for details on the PCB.

3.2.1.2 Requirements Manager

The Requirements Manager (RM) is a software tool designed to manage product requirements and evaluate the compliance of product design data with requirements. The tool allows the user to model requirements or guidelines, model the product design data structure, populate the product design data structure with product data, and evaluate to what extent the product design data meets the specified requirements. As a result of the evaluation process, the tool will provide the user with a status (Pass, Fail, Uncertain, or Untested) of the compliance of the product data with the requirements. The MO manufacturing guideline functionality is being incorporated into the RM to provide the "top level" product development team insight into manufacturing requirements apart from the MO analyses.

It is common practice for a manufacturer to document manufacturability, or producibility guidelines that delineate standard manufacturing practices and acceptable design parameters. The purpose of these guidelines is to communicate the capabilities of the manufacturing process to the product design community to ensure that new product designs are specified within manufacturing capabilities. The guidelines delineate quantitative and qualitative producibility issues.

The MO system is supporting evaluation of these manufacturing guidelines. For each guideline entry there is a related recommendation. Unlike the process selection constraints, manufacturability guideline violations may not cause alternative selection. The result could be an operation cost increase, for instance, the need for non-standard tooling, a yield loss, or a less tangible impact. These guidelines will be entered into the Requirements Manager so that they are available to the product design team along with the other requirements placed on the design. Some examples of these guidelines include: "The maximum board dimension must be less than 14 inches", "Switches must be hermetically sealed", or "If the number of leads is less than or equal to 24 the span should be 0.3 inches". See reference 9 in section 2 for details on the RM.

3.2.1.3 RAPIDS

RAPIDS is Raytheon's conceptual design and analysis workstation for Printed Wiring Boards (PWB). RAPIDS supports component placement and placement density analysis, as well as a number of other analysis functions, including automatic component insertion checking. Interfaces between RAPIDS and the PWB analysis tools for the following criteria are also provided as part of the RAPIDS tool suite:

- Manufacturing
- Post Layout Effects
- Reliability
- Thermal

At Raytheon, RAPIDS is used for conceptual design and analysis of PWB's. RAPIDS serves in the same capacity at Raytheon that many commercial CAD systems (e.g. Mentor Board Station, Racal-Redac Visula, Cadence, etc.) are used in at other companies. RAPIDS provides an Application Programmatic Interface (API) with its database. This enables RAPIDS to be easily interfaced with other systems and standards. Using RAPIDS in the MO system is inline with Raytheon methodologies, but does not exclude interfacing MO with commercially available CAD systems in the future. The key to interfacing MO with a large base of CAD systems is the utilization of the STEP standard by the commercial CAD industry. See reference 10 in section 2 for details on the RAPIDS Data Dictionary.

3.2.2 Product (STEP) Models

All data required for the MO system will be stored in STEP physical files. The reason behind the use of STEP physical files is that STEP is an emerging international standard which is getting wide spread attention as the means of exchanging data between automation systems. Access to the STEP files will be provided through the STEP Toolkit (STEP Tools Inc.). The Toolkit provides a means of reading and writing STEP entity instances through a C++ class library. This class library is currently being updated to adhere to the ISO Part 22 SDAI (Standard Data Access Interface) specification.

At Raytheon, PWB product data is stored in the RAPIDS (Raytheon's Automated Placement and Interconnect Design System) database. Two interfaces were developed to support the transition of PWB product data to and from STEP physical files.

Generating the STEP physical file is facilitated by the *RAPIDS to STEP* interface which maps RAPIDS data items into instantiated STEP entities. An information model using the EXPRESS information modeling language was created based on the RAPIDS database. The EXPRESS information model was compiled using the STEP Tools *express2c++* compiler, which generated a STEP schema and a C++ class library. The class library consists of methods for creating and referencing persistent instances of the STEP entities which are stored in a ROSE database. The STEP schema is used by the STEP Tools *STEP filer* for reading and writing the STEP physical file.

The MO system uses the STEP data directly, as well as, for information exchange between the members of the product design team. For demonstration purposes, we will have the top level team using RAPIDS. This is not a requirement for using the MO system. The only requirement is that the top level team and the lower level teams be capable of creating, exchanging and using the STEP physical file.

The Manufacturing Team passes back a consolidated manufacturing position to the product design team. To aid in the generation of a consolidated position, conflict resolution and design merging must be supported. This is done using the STEP Toolkit from STEP Tools Inc. The *diff* tool reads two versions of a design and creates a delta file. The *difference report generator* reads the difference file and the original design, and presents each STEP entity and its attributes with the original values and its change state clearly marked with an asterisks.

Once the conflicts of the Manufacturing team members have been resolved, design versions are merged using the STEP Tools *sed* tool. The *sed* tool reads the delta file created by the *diff* tool and updates the original design version. This updated version of the design will be transferred back to the top-level product team as the Manufacturing Team's consolidated position.

3.2.3 Process Models

The key to performing manufacturability analysis is to characterize the fabrication and assembly processes. In MO, this characterization is implemented as a manufacturing process representation and selection algorithm. Basing manufacturing cost analysis on a detailed description of the process provides visibility into the relationship between the design attributes and the manufacturing process. This allows the engineer to focus on manufacturing cost drivers and their causes. By characterizing the process in this manner, the manufacturing engineer will be able to review the process which will be used to produce the product and be readily able to consider alternative manufacturing processes and their consequences.

Following this logic, it makes sense to capture the expert's process planning knowledge into a process selection model so that the relationship between the product entities and the process selected to fabricate the product is explicitly defined. This does not mean that there is a one to one relationship between the design entities and the process steps. In some areas, such as PWB, the design may be implemented using different technologies, each of which implies a certain process, such as surface mount versus through-hole technology. In other cases, there are multiple processes that can be used to produce the same entity. This is most prevalent in the metal fabrication (machining) area where often a number of processes (investment casting, milling) are capable of producing the part.

There are two development challenges: building a data schema to represent the manufacturing process such that it can be used for selection and costing, and building a selection logic algorithm that adequately represents the planning logic employed by expert process planning.

Normally in a manufacturing plant, the overall process for a given discipline is known and recorded in the form of a flowchart. This flowchart is a block diagram listing of each and every process within that discipline. The order of those operations is structured so that it is the default ordering of how products flow. If a process gets repeated, it generally shows up in each repeated point in the flow chart. These flowcharts usually employ a rudimentary decision logic scheme. As such it represents the available processes in a pick list fashion. Pictured in figure 3.2-2 is a typical manufacturing process flow for printed wiring boards.

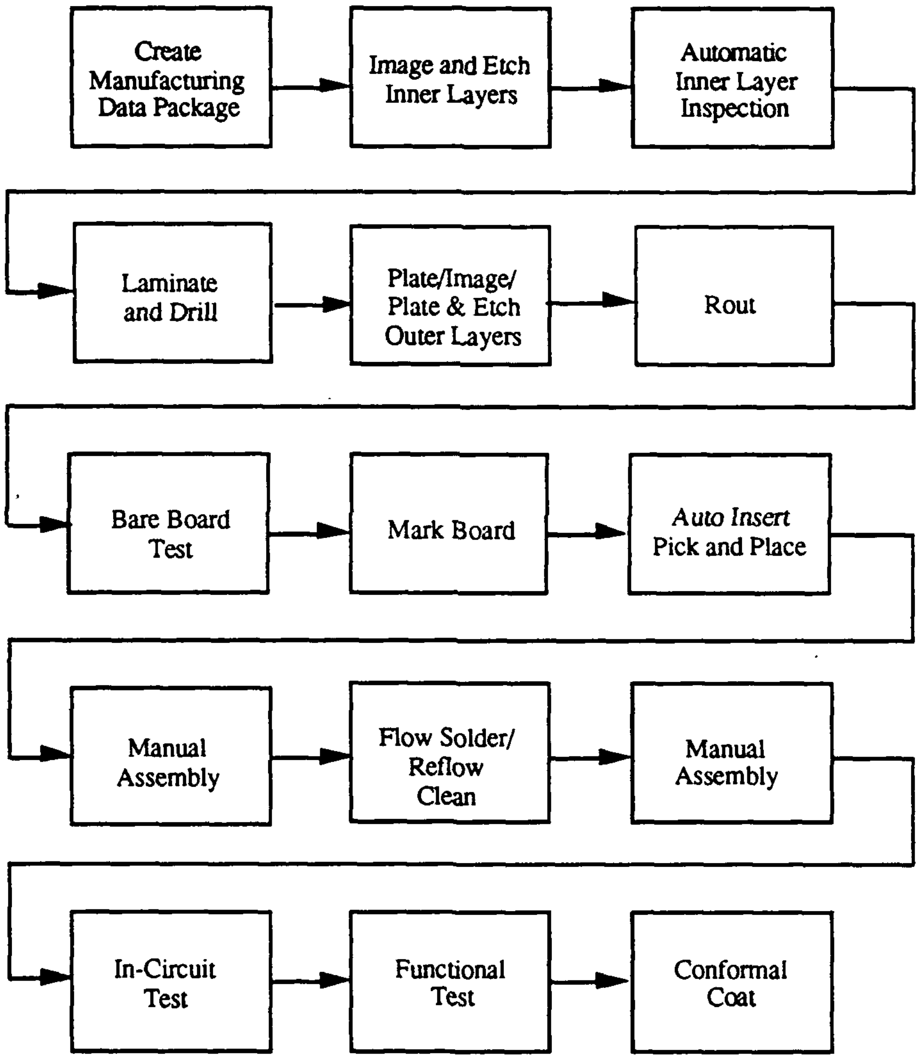

Create Manufacturing Data Package
Image and Etch Inner Layers
Automatic Inner Layer Inspection
Laminate and Drill
Plate/Image/ Plate & Etch Outer Layers
Rout
Bare Board Test
Mark Board
Auto Insert Pick and Place
Manual Assembly
Flow Solder/ Reflow Clean
Manual Assembly
In-Circuit Test
Functional Test
Conformal Coat

Figure 3.2-2 Printed Wiring Board Manufacturing Flow

The logic representation method that Raytheon is developing for this task is based on prior work in process selection. The model is a hybrid and/or dependency graph and rule based processing system. The and/or dependency graph representation was selected because it allows the system to display the basic sequential and concurrent flow of the process in a presentation format where the manufacturing engineer can visually see other process(es) that a selected process is dependent on. The dependencies inform the user of the basic flow of the overall process while letting the user plan at various levels of abstraction. These levels include the process, an organized group of manufacturing operations sharing characteristics, the operation, a common unit of work that is performed on the part, and the resource, which is the mechanism required to perform the operation. By defining the levels as we have, a hierarchical planning strategy is enabled. Using this schema, we can reason about alternative processes, plan the operation flow within the selected process, and then detail the individual resources of that operation, such as set-up and run time standards.

The reasoning process is guided by the representation of the dependency graph which sets the initial search evaluation order, and the rule processing mechanisms. The rules are attached to individual process nodes in the graph. These rules are used to evaluate the node. The purpose of the evaluation is to cause selection of the node. First, any parent processes that the process under evaluation is dependent on must be satisfied. If satisfied, the rules attached to the process are evaluated to see if it is applicable to manufacturing the product part. Operations are stored to form the overall manufacturing process sequence. Each operation in the process sequence is evaluated for its requirement of resources in order to estimate the manufacturing process cost.

The system will also have the ability to store alternative models of a particular process. This capability will allow the process engineer the ability to explore alternative process approaches and plan process improvements. Figure 3.2-3 illustrates a sample assembly dependency graph for through-hole circuit cards.

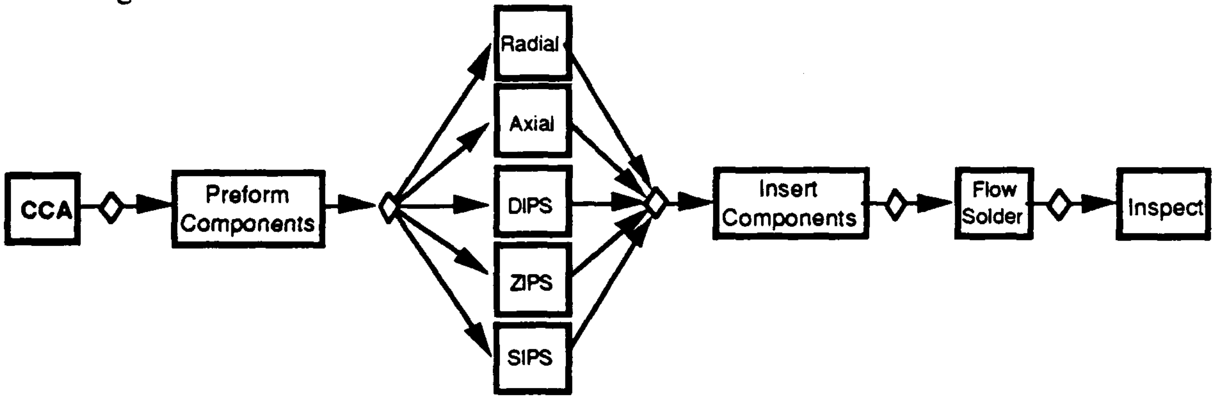


Figure 3.2-3 Sample CCA Process Dependency Graph

3.2.4 Manufacturing Analyzer

There are three capabilities provided in the Manufacturing Analyzer module: process analyzer, yield and rework analyzer, and cost estimator. The sub-sections to follow describe each capability.

3.2.4.1 Process Analyzer

The Process Analyzer provides the capability to select or determine the process sequence required to manufacture the product design based on a particular process model. The manufacturing process is represented by three levels of abstractions: process, operation and resource. The process is an organized group of manufacturing operations, the operation is a common unit of work that is performed on the part, and the resource is the mechanism required to perform the operation. The process model for the MO system is designed as an "and/or dependency graph" made up of selectable manufacturing processes. Each process node in the graph can be connected to process(es) at a higher level and/or lower level in the graph. A list of applicable operations and resources can be attached to the process in the dependency graph. Each operation has applicable yield and rework rates attached. Refer to Section 6.1 for the details of the process model schema.

The Analyzer will select the applicable process(es) on a level by level basis using the selected process model. First, the and/or prerequisite of the parent(s) of the process must be satisfied. If satisfied, the rules attached to the process are evaluated to see if it is an applicable process. The selected process and corresponding product design entities will be added to a dependency analysis graph. Figure 3.2-4 illustrates a resulting high level process analysis graph for a circuit card.

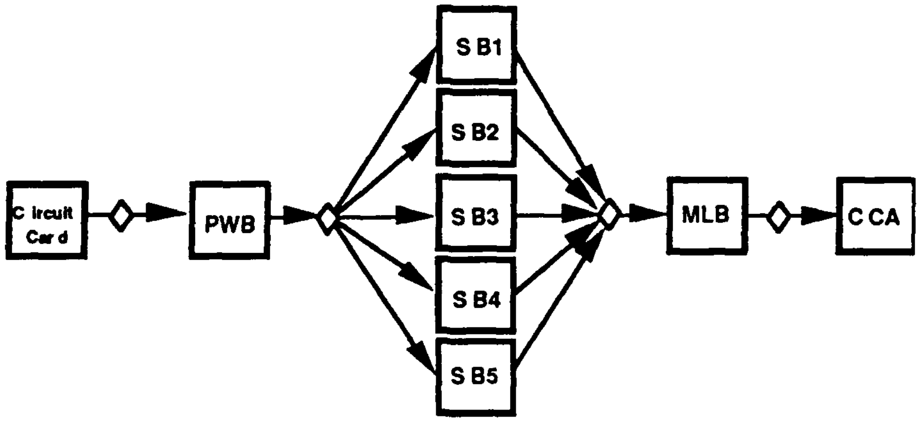


Figure 3.2-4 High Level Process Analysis Graph

3.2.4.2 Yield & Rework Analyzer

The yield and rework analyzer provides the capability to calculate yield and rework rates for the selected processes associated with a product design. This part of the analysis calculates the yield and/or rework rate on an operation level within the process. The rate will be calculated based on the design entities influence on the operation. The yield and/or rework rate for each design entity/entities associated with an operation is calculated through the evaluation of a rule, which has a corresponding equation attached. If the rule evaluates to true, then the equation will be calculated to provide the yield or rework rate. The rate equations may include references to the existence, value, or quantity of product design entities. An example yield rule and corresponding rate attached to an operation is as follows:

<u>**Yield Data:**</u>

Design Features Rule	Scrap Rate
aspect ratio < 5.0 & aspect ratio > 4.0	0.05000
aspect ratio <= 4.0 & aspect ratio > 2.0	0.02000

The total yield rate for an operation is calculated using the statistical probability of each design entity scrap rate to each other by calculating the independent events. The total rework rate for an operation is calculated by summing up the results of each rework occurrence.

3.2.4.3 Cost Estimator

The cost estimator calculates the recurring manufacturing cost for each operation in the process sequence. The following calculations are performed:

- Labor standards for each resource attached to an operation are calculated for setup and run time utilization. The value for each is calculated through the evaluation of an equation which may include reference to the existence, value, or quantity of design entities in the product data. Each resource has an associated cost in terms of an appropriate measure. For example, a labor resource will have an associated cost in terms of dollars per time unit.

- Estimated ideal cost for each operation is calculated from labor standard values multiplied by the wage rate of the labor grade or bid code of the resource(s) performing the operation, and the production efficiency value for that operation.

- Rework operations are calculated based on the rework rate determined by the yield and rework analyzer multiplied by labor standards of the resources for the rework condition. The labor grade wage rates and production efficiencies would then be applied.

- For each operation, the estimated actual cost is calculated by multiplying the estimated ideal cost by the number of units processed, including both good and scrapped units. The number of units processed by each operation are calculated from the value of the required good units at the subsequent operation divided by the yield at the operation under evaluation. For example, if the desired production quantity is 100 boards and operation 1 has a scrap rate of 5%, then the quantity of required units to being operation 1 with is 105.

- The total estimated ideal cost and total estimated actual cost for each sequence of processes are calculated by summing the individual operation cost of each process. The estimated actual cost for a good unit is calculated by dividing the total estimated actual cost for the process by the number of good units produced.

3.2.5 Manufacturing Advisor

The manufacturing advisor provides the capability to view the results produced by each process participating in an analysis. The advisor includes the following capabilities:

- A mechanism for selecting one or more manufacturing analyzer runs for comparing and/or displaying the results.

- Graphical capabilities (i.e. line, bar, stacked bar and pie charts) for comparing and displaying the process, yield, rework, or cost versus a processes or operations for one or more manufacturing analyzer runs.

- A reporting capability which prints analyzer results to the screen or file for one or more runs including process sequence, yield and rework, and cost.

- The capability to summarize design entities causing *manufacturing guideline violations* (interface to the RM) across multiple processes. Report recommendations on these guideline violations.

- A final manufacturing summary report, identifying cost drivers, for each process contributing to a multi-process analysis for a given design database after completion of the manufacturing optimization process.

Provided below in figure 3.2-5 is a sample of the type of graphical display the user would see for yield versus process.

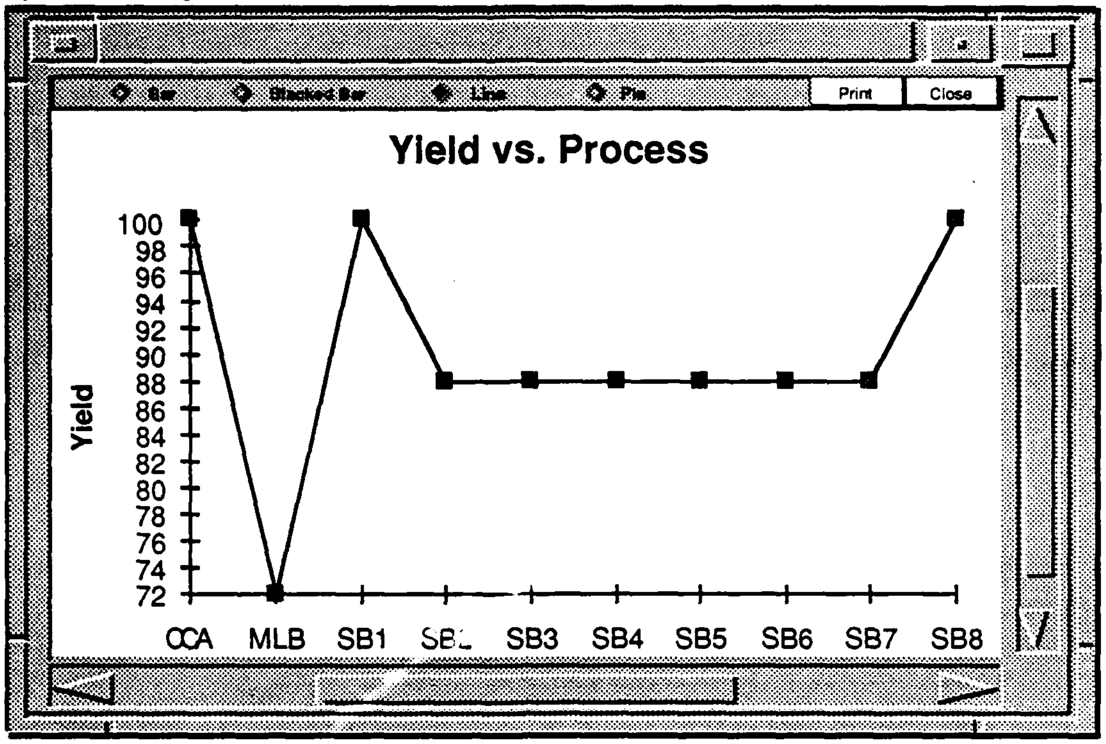


Figure 3.2-5 Sample Yield versus Process Display Graph

3.2.6 Process Modeler

The process modeler provides the capability to model the selection logic of the manufacturing process. The process model is decomposed into a graph of process nodes. Each process node consists of the following:

- Selection rules - If these rules are satisfied and this nodes dependencies are satisfied, then the process node is included in the total process analysis model.
- Dependencies (parent process nodes) - For the node under consideration to be selected, all of the process nodes that it is dependent on must be satisfied. An AND/OR flag is stored as each node in the graph. If the flag is set to AND then, the node is dependent on all of its parents. If the flag is set to OR, then the node is depending on only one of its parents to be satisfied.
- Operations - At each process node there is a list of operations that are performed. Each operation is annotated with an associated yield rate, rework rate, and its usage of resources.

- Resources - At each process and operation node there is a list of resources attached. A resource is any facility, person, equipment, or consumable material used in the manufacturing process.

The MO system will allow the manufacturing specialists to capture and maintain multiple copies of process data models through a set of utilities. The utilities will provide the model developer with the tools necessary to graphically build and view the process logic dependency graph, selection rules, yield/rework, labor standards, and resources. Through the use of these utilities, the process team will have the ability to modify the process model data, to explore alternative process approaches and plan process improvements, and then analyze the effects of these changes on the product cost. The user interface will consist of pull down menus and pop up forms to allow adding, copying, moving, deleting, editing, and printing of the processes in the dependency graph. Pictured below in figure 3.2-6 is the main user interface window for the Process Modeler with a sample process model displayed.

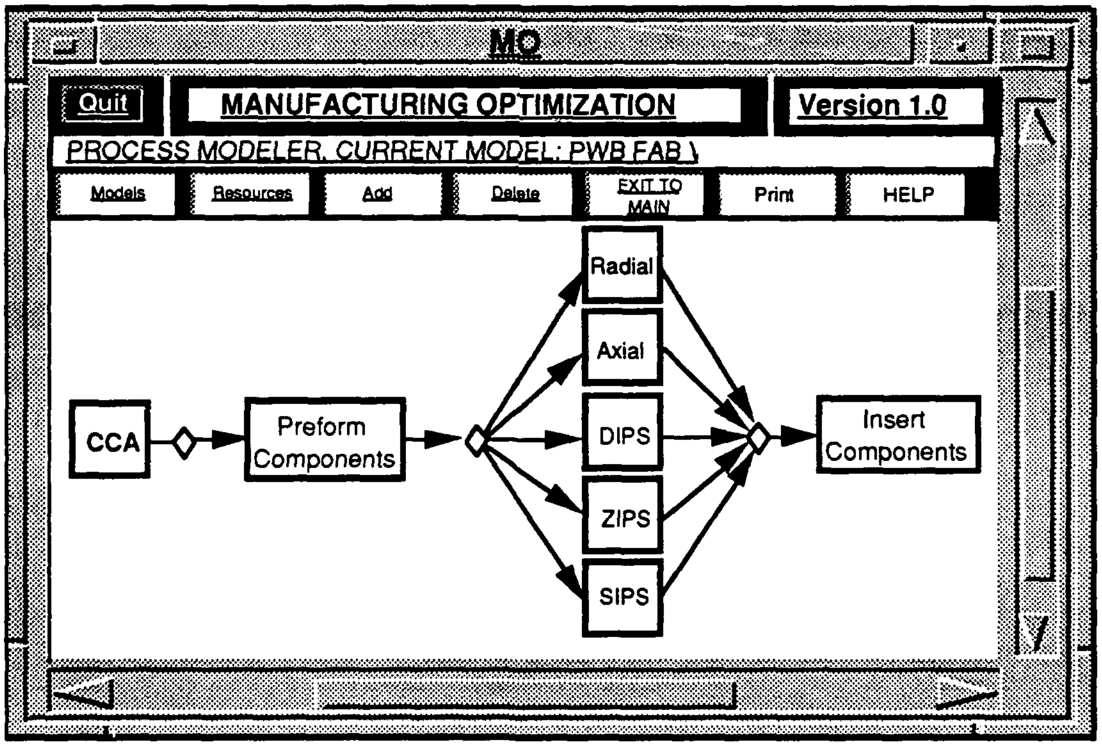


Figure 3.2-6 Process Modeler User Interface Window

4. User Interface Design

The main user interface window for MO provides access to the various modules within the system, including the product and process STEP files, the manufacturing analyzer, the manufacturing advisor, the process modeler, anu system help. Figure 4-1 depicts the MO main window.

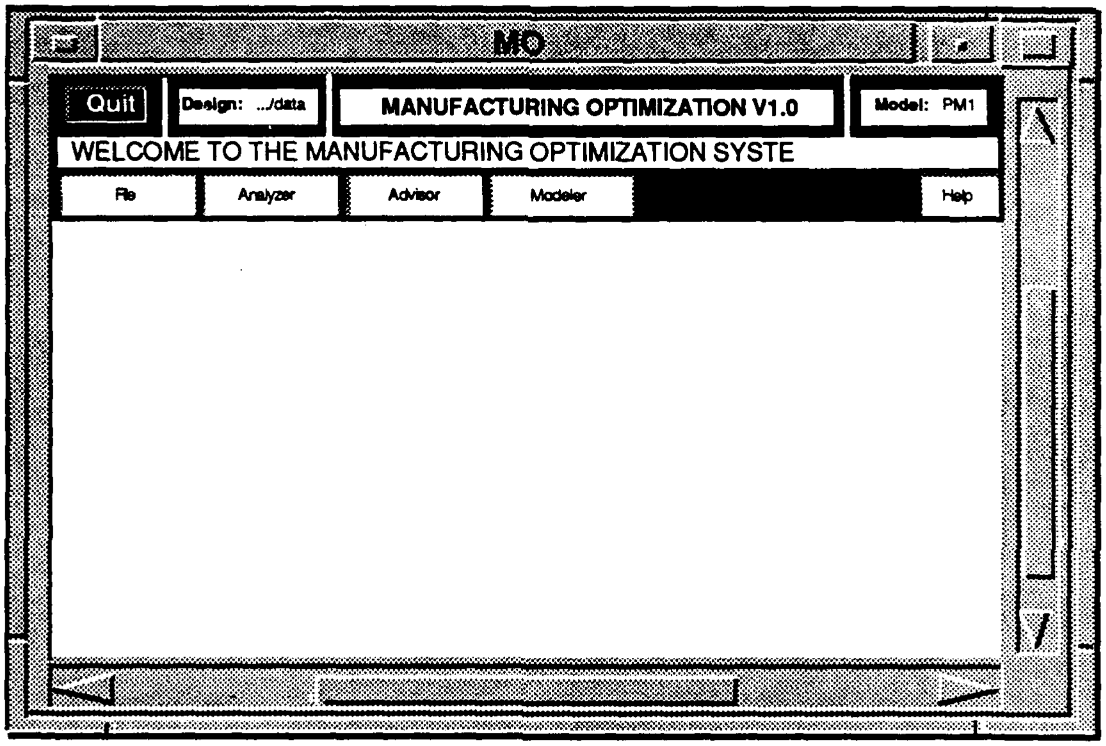

MO
Quit
Design: .../data
MANUFACTURING OPTIMIZATION V1.0
Model: PM1
WELCOME TO THE MANUFACTURING OPTIMIZATION SYSTE
File
Analyzer
Advisor
Modeler
Help

Figure 4-1 MO Main Window

4.1 File Menu

The File menu provides a means to select and edit the product and process data and provides access to two translators. Rapids2Step translates PWB design data from a Raytheon propriety format to STEP. Step2Rapids translated a PWB design from STEP to a Raytheon propriety format. Figure 4.1-1 illustrates the MO main window with the File menu pulled down.

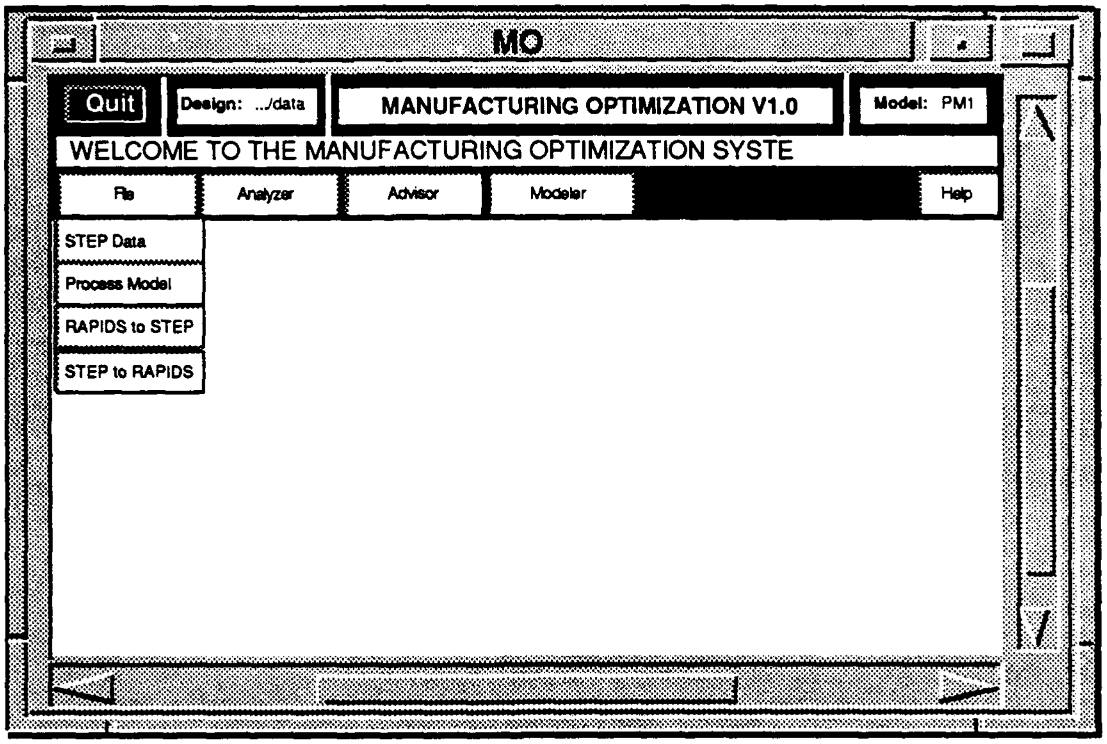


Figure 4.1-1 File Options

4.1.1 Product/STEP Data Selection

MO allows the user to select a product/step data file for analysis, or to edit a step file in the Step Toolkit Editor. When the Step Data button is selected, figure 4.1-2 is displayed. A user performs a selection for choosing a design database to analyze or a process model for use during analysis. When the Edit button is selected the STEP Editor from STEP Tools Inc. is invoked with the selected STEP file loaded. The STEP Editor enables the user to add, delete, and modify STEP entity instances.

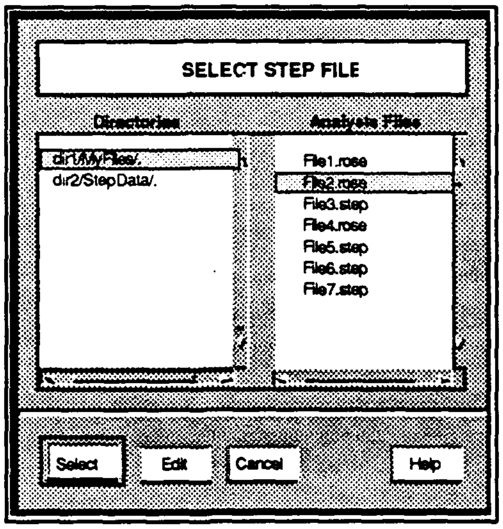


Figure 4.1-2 Product Data Selection/Edit Menu

4.1.2 Process Model Selection

The MO system allows the manufacturing engineer to capture and maintain multiple copies of process data models through a set of utilities. Through the use of these utilities, the process team has the ability to modify the process model data, to explore alternative process approaches and plan process improvements, and then analyze the effects of these changes on the product manufacturing cost. Figure 4.1-3 shows the user interface provided for process model selection.

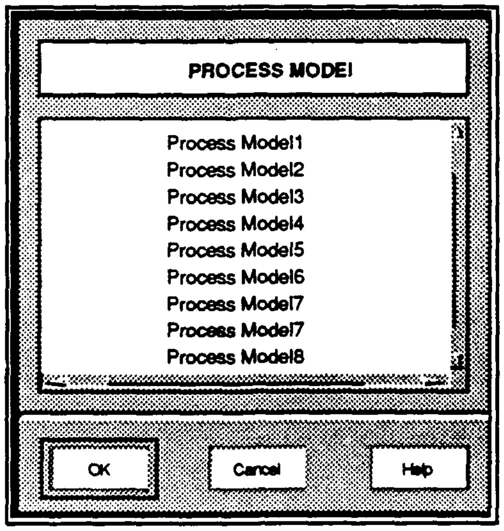


Figure 4.1-3 Process Model Selection Menu

4.1.3 RAPIDS to STEP Translator Interface

Rapids2step is a C++ application that utilizes the ROSE database and tools developed by STEP Tools Inc. The program reads the RAPIDS Structured and Library Databases (RSD/RLD) using the RAPIDS Procedural Interface. For each RSD and RLD file type, a function of the form rsd_read_*xyz*_record() is called where *xyz* is the type of record requested. The function returns a pointer to a structure containing the data of the next unread record from that particular RAPIDS database file. A persistent STEP object of the appropriate class is generated using the ROSE *pnew* instruction. The attributes of the object are populated from the appropriate data in the structure returned by the rsd_read_*xyz*_record call and the object is added to a persistent list of objects of that type. Once all of the records have been read from the RSD and RLD databases and the corresponding STEP object lists have been created, the STEP file is created and the STEP objects are written to it by the ROSE STEP filer. See figure 4.1-4 for an illustration of rapids2step process.

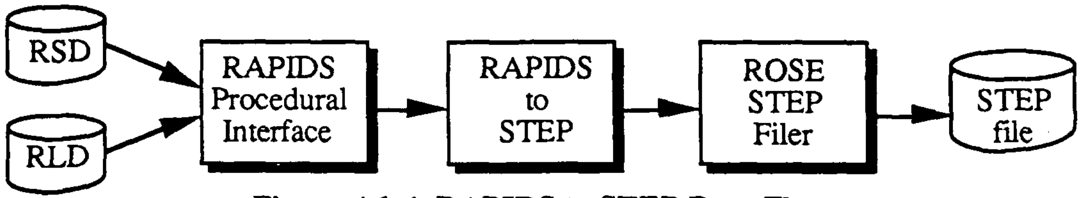


Figure 4.1-4 RAPIDS to STEP Data Flow

The MO system provides the user with an interface to the rapids2step translator. The interface is shown in figure 4.1-5.

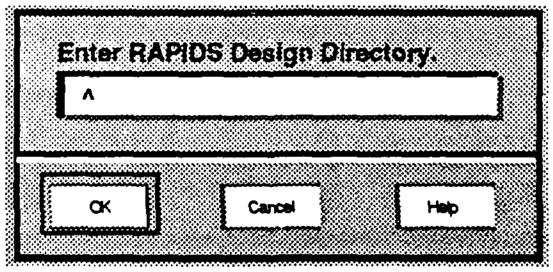


Figure 4.1-5 RAPIDS to STEP Form

4.1.4 STEP to RAPIDS Translator Interface

Step2rapids is also a C++ application that utilizes ROSE and tools developed by STEP Tools Inc. The program reads a STEP file conforming to the EXPRESS schemas developed as part of this project. The ROSE STEP filer is used to read the STEP file into instances of classes created by the express2c++ compiler. Since these STEP objects were stored in persistent lists, these lists are still available in the STEP file. The file structures for the RAPIDS RSD and RLD are created. Each of the STEP object lists is traversed and for each object in the list an appropriate C structure corresponding to the RAPIDS procedural interface is created and its fields are populated with the values of the corresponding attributes of the STEP object. A call is then made to the appropriate rsd_write_*xyz*_record of the RAPIDS procedural interface where *xyz* is the type of RSD record to be written. See figure 4.1-6 for an illustration of the step2rapids process.

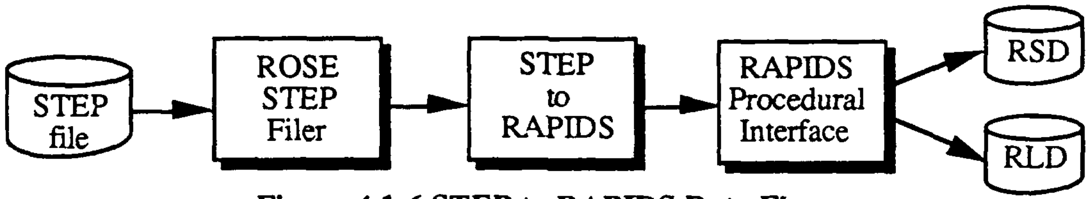


Figure 4.1-6 STEP to RAPIDS Data Flow

The MO system provides the user with an interface to the step2rapids translator. The interface is shown in figure 4.1-7.

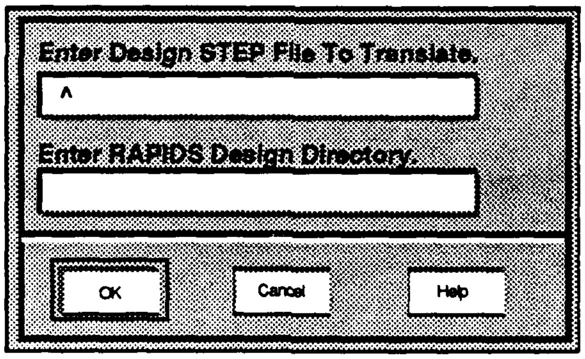

Enter Design STEP File To Translate.
^
Enter RAPIDS Design Directory.
OK
Cancel
Help

Figure 4.1-7 STEP to RAPIDS Form

4.2 Analyzer Form

The MO system provides the user with the ability to perform a manufacturability analysis based on a selected manufacturing process model versus a particular product design database through the analyzer button on the main window. The analyzer has five options to choose from: process flow, yield, rework, labor time, and cost. Process flow selects the appropriate processes required to build the product based on the selected process model. Yield and rework calculate the overall yield and rework rates of a process on an operation by operation based on the selected process flow. Labor time calculates the ideal time to perform the processes on an operation by operation basis. The yield rates are incorporated to project the estimated actual times. Cost utilizes the ideal and estimated actual labor times by multiplying them with the resource(s) labor rate(s) to obtain the ideal and estimated actual cost of each process and operation, as well as, the cost of the entire part. When a user selects the analyzer button, he/she must at least select the process flow option since this is the input to all other analysis options. Figure 4.2-1 depicts the form which is displayed when the analyzer button is selected off the Main Window. The user can then select the type(s) of analysis to be performed.

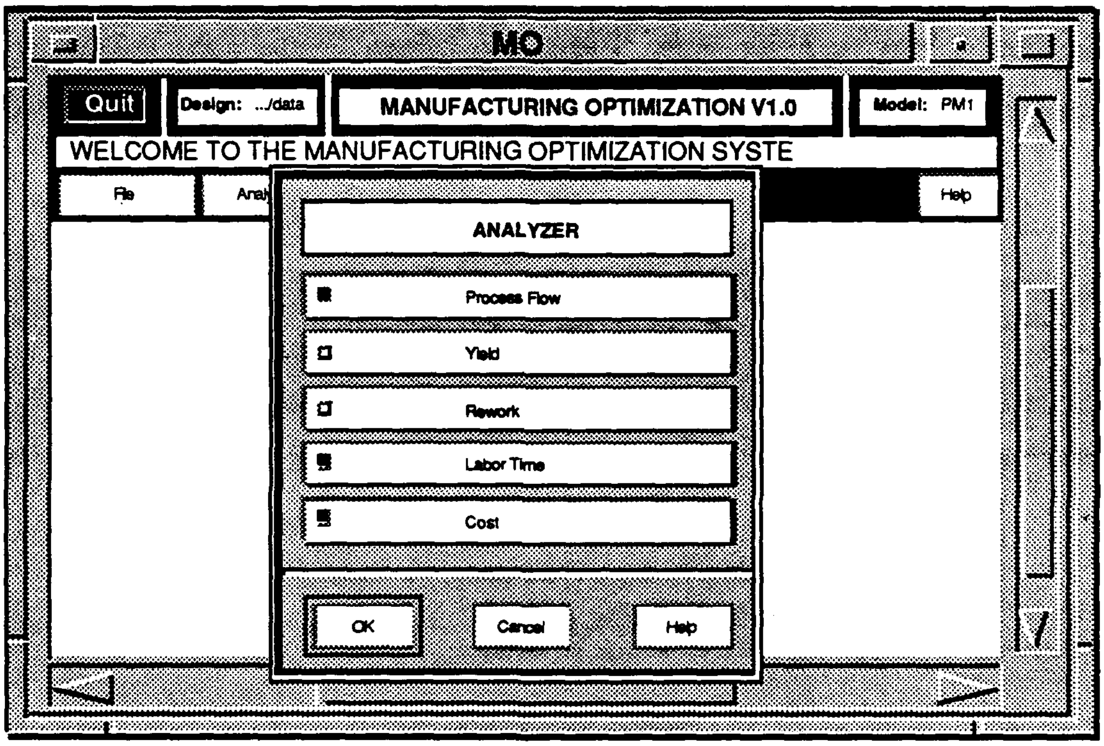


Figure 4.2-1 Manufacturing Analyzer Form

4.3 Advisor Window

The Manufacturing Advisor module provides the capability of viewing the analyzer results. The user can select analysis runs to view. The user can display process, quality, or costing results as graphs, and can also view complete analysis data to the screen or to file in report format. Figure 4.3-1 illustrates the Manufacturing Advisor window which is displayed when the user hits the Advisor button on the Main Window.

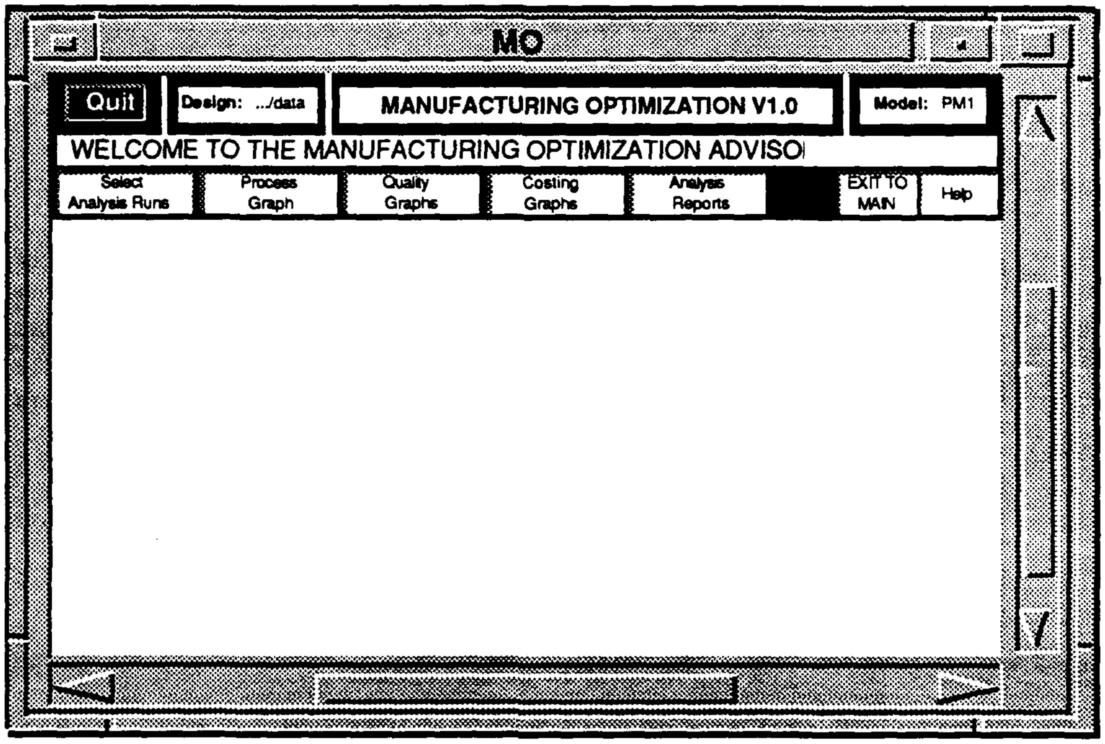
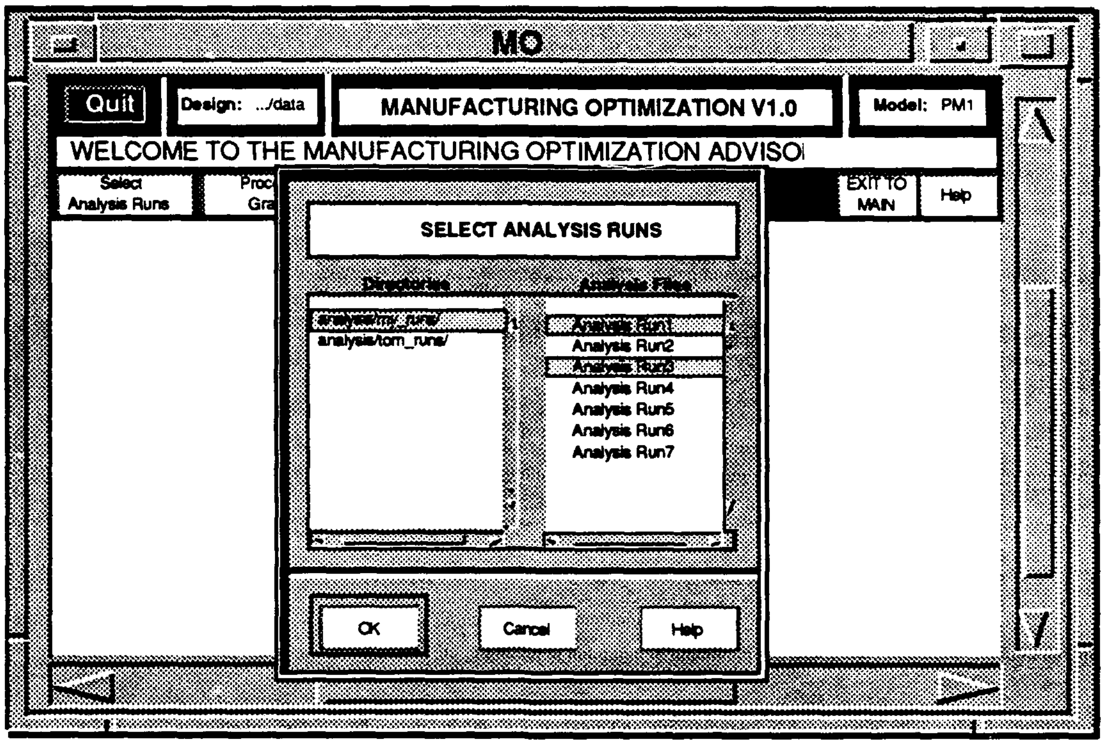


Figure 4.3-1 Manufacturing Advisor Window

4.3.1 Select Analysis Runs

The MO system supports viewing of one or more analysis runs so that the user can visually see the results, as well as visually compare analyses. Figure 4.3-2 illustrates the form that is displayed when the user hits the Select Analysis Runs button on the Manufacturing Advisor Window. The user can select the run(s) which he/she wants to view. The default selection is the analysis results which corresponds to the last analyzer run performed.



Design: .../data

Model: PM1

WELCOME TO THE MANUFACTURING OPTIMIZATION ADVISO

Directories	Analysis Files
analysis/my_runs/	Analysis Run1
analysis/tom_runs/	Analysis Run2
	Analysis Run3
	Analysis Run4
	Analysis Run5
	Analysis Run6
	Analysis Run7

Figure 4.3-2 Select Analysis Runs Form

4.3.2 Process Graph Display

When the Process Graph button is chosen, the selected analysis run(s) process flow is graphically displayed. The graph can be expanded or compressed by the user to different levels of abstraction by selecting the diamond shapes on the drawing. Each process is displayed as a square button with the name of the process shown inside. Figure 4.3-3 illustrates the resulting process flow graph for one set of analysis results.

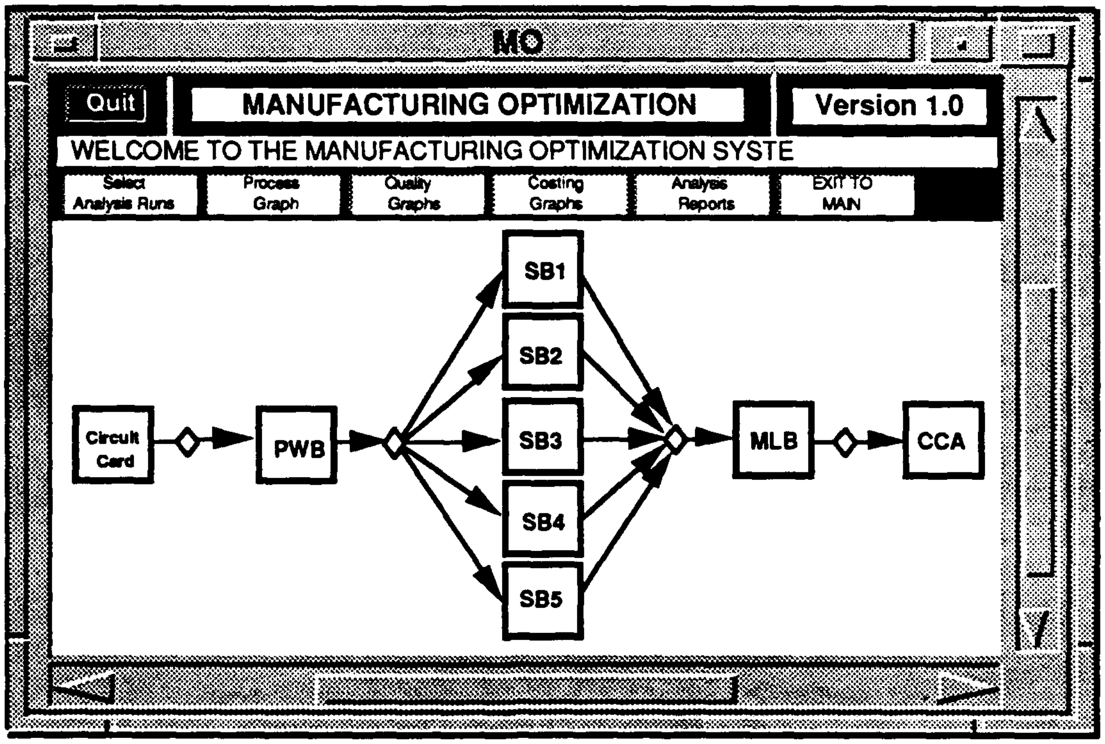
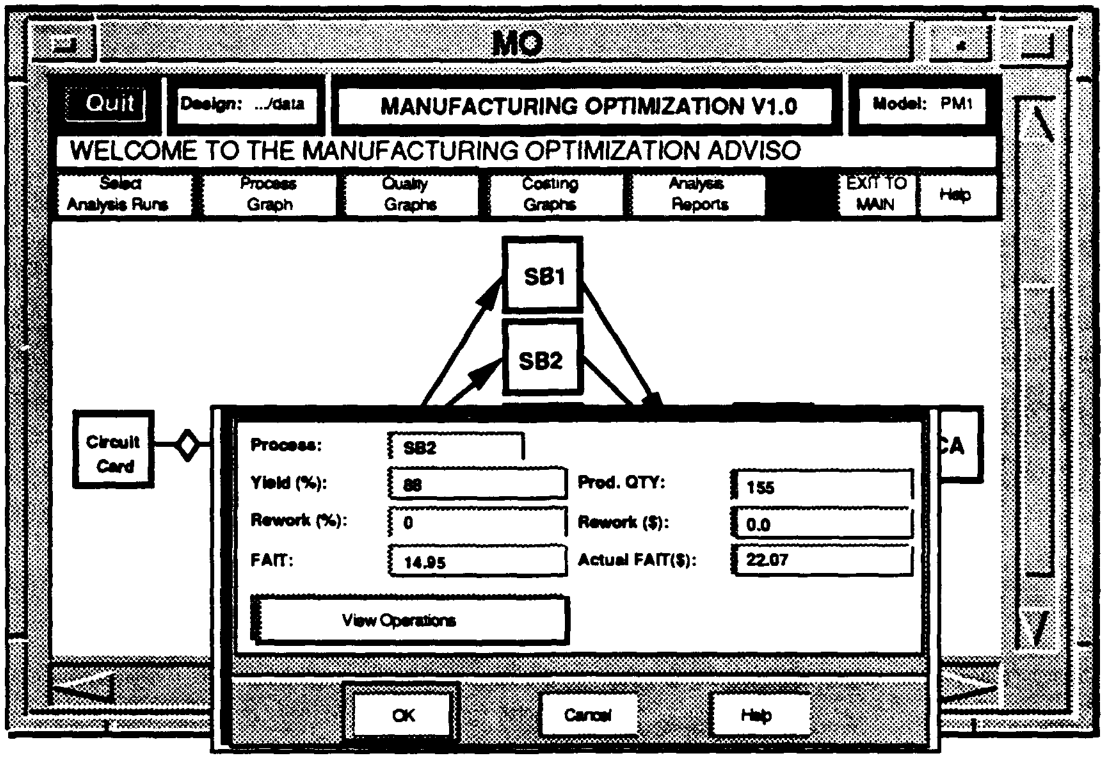


Figure 4.3-3 Process Graph

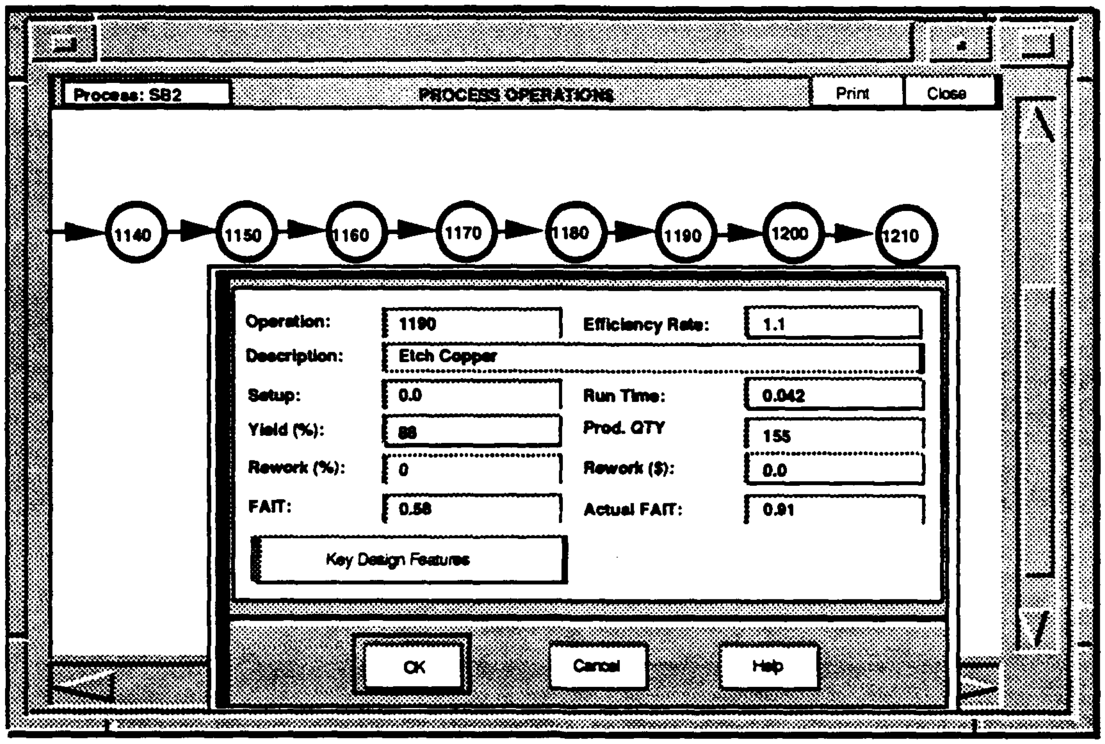
The user can then chose to select a process button on the graph in order to see the analysis details for that process including: process name, yield and rework percentage, production quantity, rework cost, ideal FAIT (fabrication, assembly, inspection, and test cost), and the estimated actual FAIT. If that process has a set of operations attached to it, the user can also request to view them graphically. Figure 4.3-4 illustrates the form that is displayed when a particular process is selected.



Process: SB2
Yield (%): 88
Prod. QTY: 155
Rework (%): 0
Rework ($): 0.0
FAIT: 14.95
Actual FAIT($): 22.07

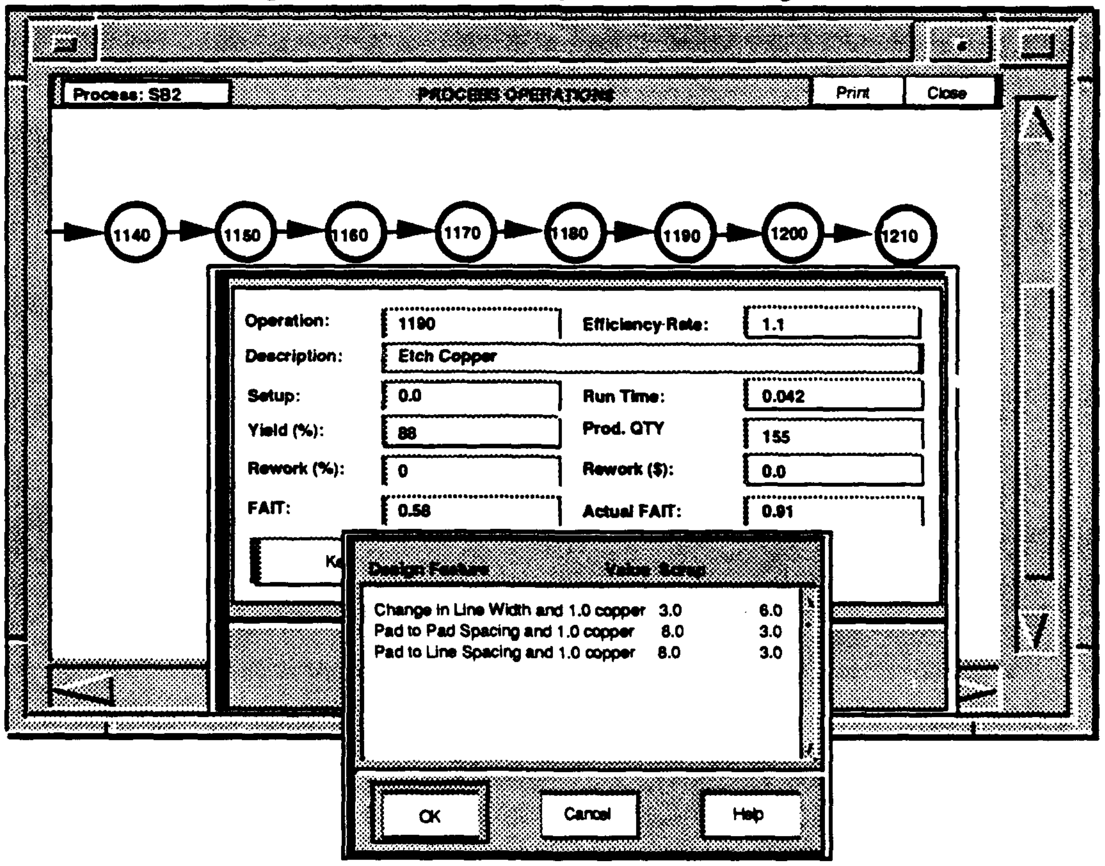
Figure 4.3-4 Process Results Viewing Form

If the user chooses to view the available list of process operations, they will be displayed in a separate window. Each operation is displayed as a circle button with the name of the operation shown inside. The window identifies the process that the operations belong to, and allows the user to print and close that window. The user can then decide to select an operation button on the graph to see the analysis details for a particular operation. The details include operation name, setup and run time, efficiency rate, yield and rework percentage, production quantity, rework cost, ideal FAIT (fabrication, assembly, inspection, and test cost), and the estimated actual FAIT. If that operation has a set of design entities (features) contributing to the quality of the board, the user can request to look at the details. Figures 4.3-5 and 4.3-6 illustrate the process operations viewing window, operation details form, and the design entity form for displaying the quality issues.



Operation:	1190	Efficiency Rate:	1.1
Description:	Etch Copper		
Setup:	0.0	Run Time:	0.042
Yield (%):	88	Prod. QTY	155
Rework (%):	0	Rework ($):	0.0
FAIT:	0.58	Actual FAIT:	0.91

Figure 4.3-5 Process Operation Viewing Form



Operation:	1190	Efficiency Rate:	1.1
Description:	Etch Copper		
Setup:	0.0	Run Time:	0.042
Yield (%):	88	Prod. QTY	155
Rework (%):	0	Rework ($):	0.0
FAIT:	0.58	Actual FAIT:	0.91

Design Feature	Value	Setup
Change in Line Width and 1.0 copper	3.0	6.0
Pad to Pad Spacing and 1.0 copper	8.0	3.0
Pad to Line Spacing and 1.0 copper	8.0	3.0

Figure 4.3-6 Operation Design Entities Viewing Form

4.3.3 Available Quality Graphs

The MO system provides for graphically displaying the quality results associated with analysis runs including graphs for yield, rework, and production quantity. Figure 4.3-7 illustrates the Advisor Window with the Quality Graphs menu pulled down.

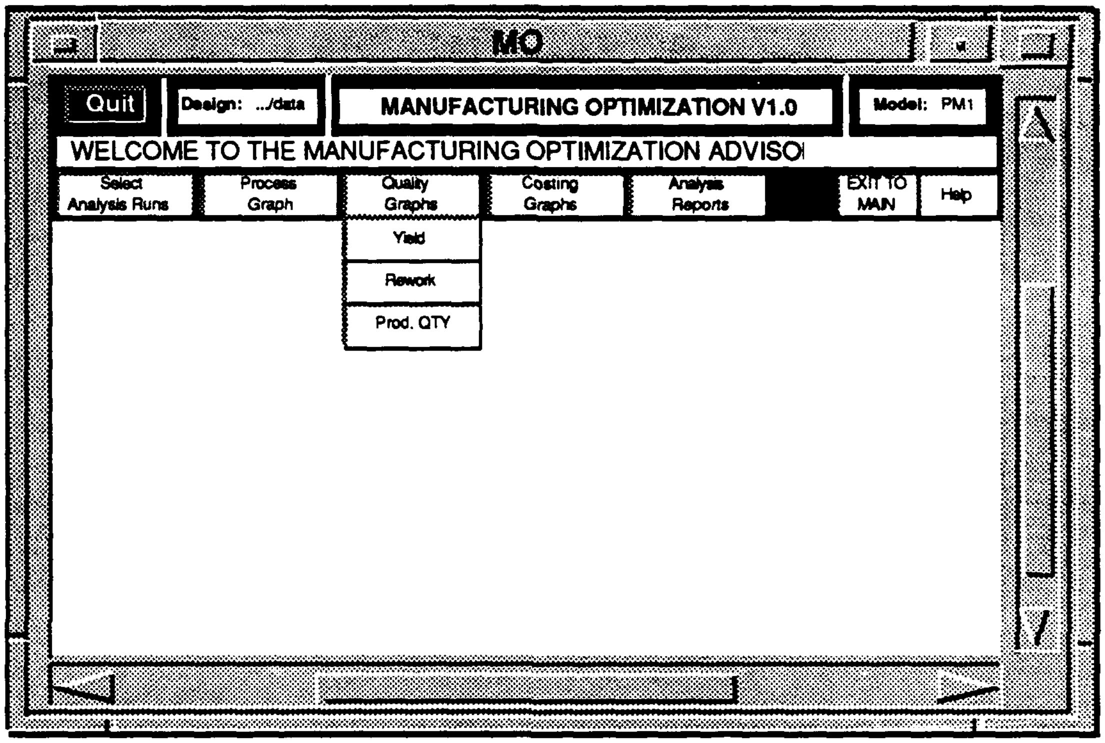


Figure 4.3-7 Quality Graphs

4.3.3.1 Yield Graphs

The type of yield quality graphs available are yield rates versus processes and yield rates versus operations associated with a particular user selected process. Figure 4.3-8 illustrates the pull out menu for the quality yield graphs. The graphs are displayed in a separate window where the user can select to display the data as a bar, stacked bar, line, or pie chart. The yield defaulting display will be a line chart. A sample line graph of yield versus process is depicted in Figure 4.3-9.

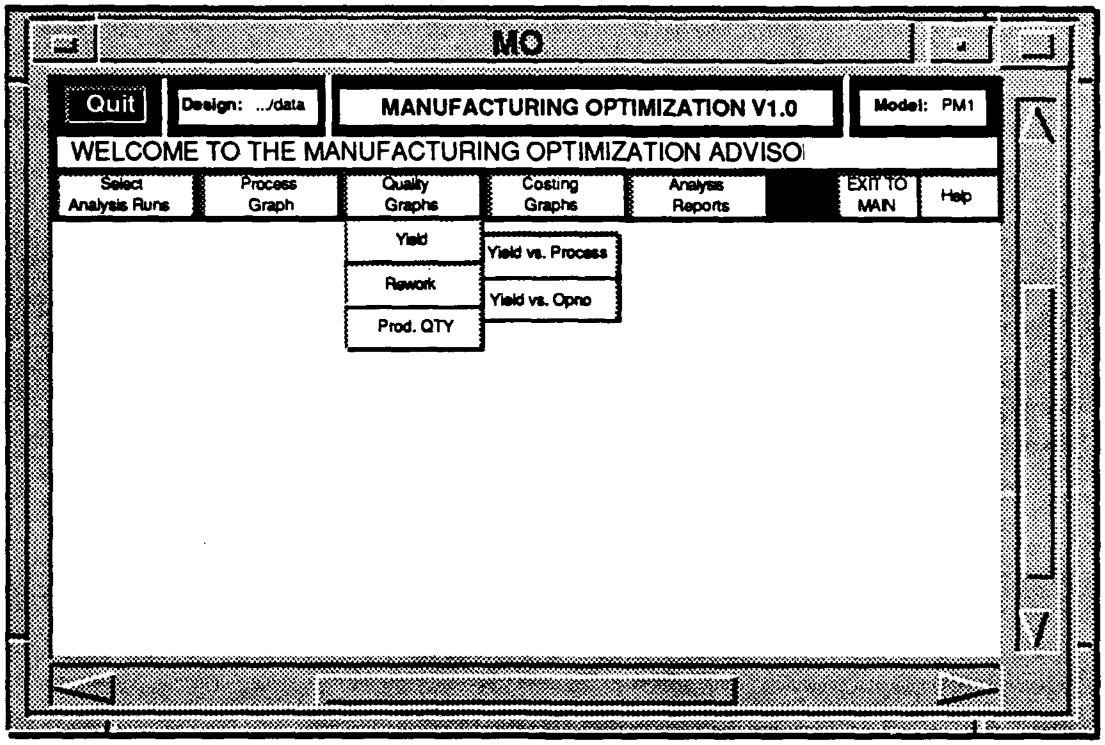


Figure 4.3-8 Yield Graphs

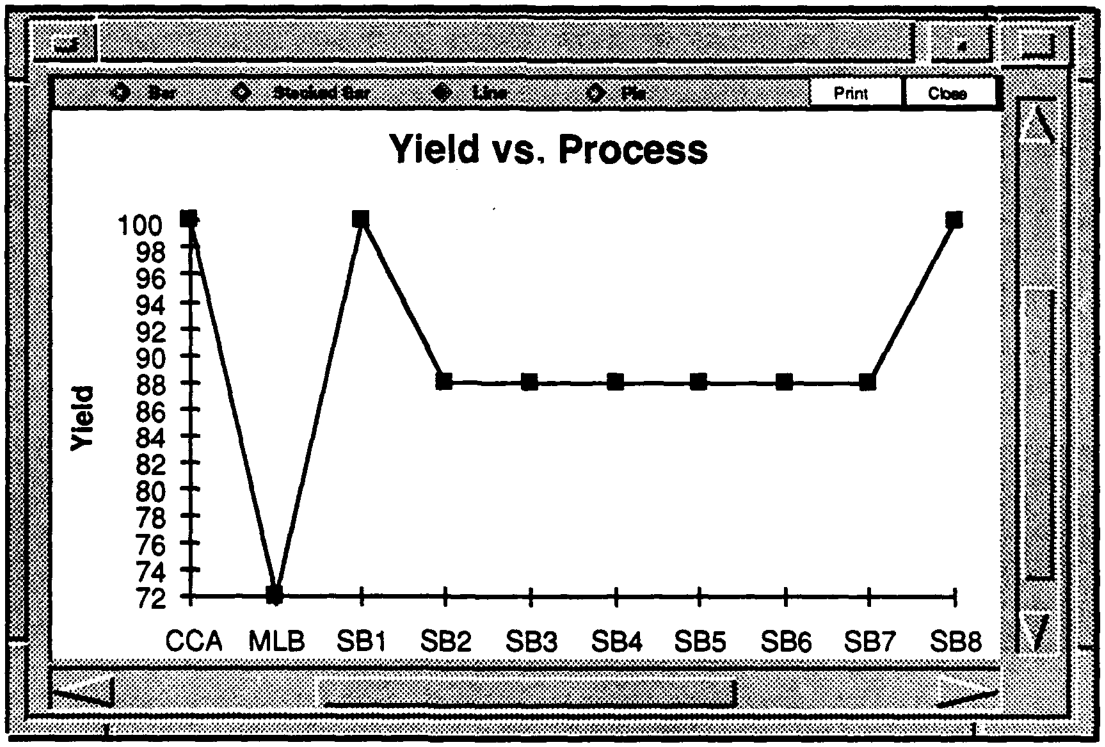


Figure 4.3-9 Yield versus Process Graph

If the user wants to view yield versus process operations figure 4.3-10 is displayed so that they can select from the available processes to identify the set of process operations. Figure 4.3-11 illustrates the resulting yield versus operations line chart for the selected process.

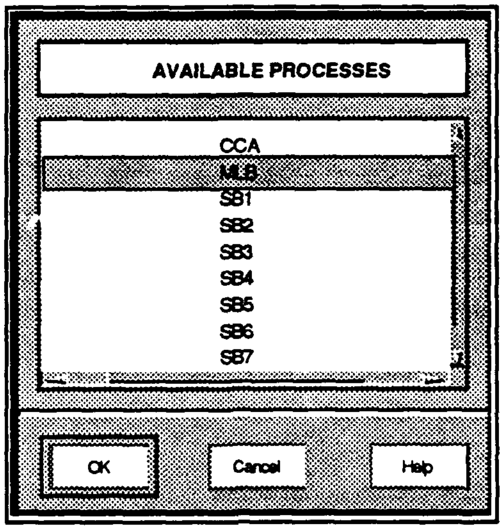


Figure 4.3-10 Available Processes to Select

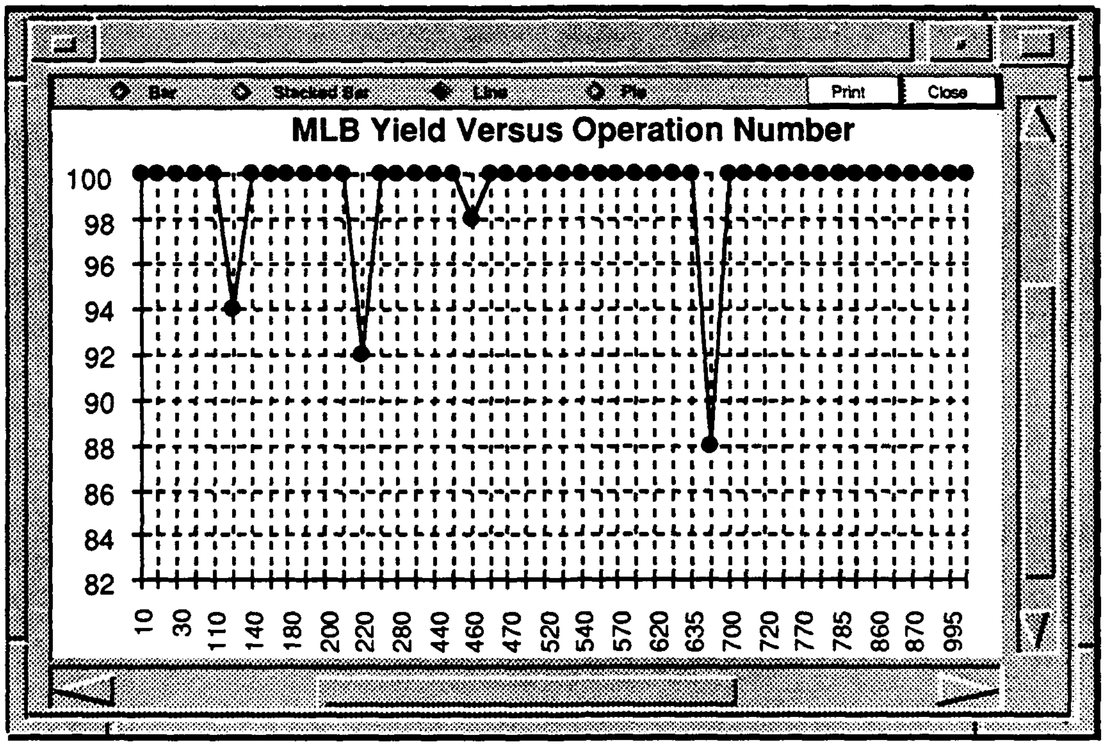


Figure 4.3-11 Yield versus Operations Graph

If the user wants to compare the yield rates versus process for two runs, he/she would select the analysis runs from the form under the Select Analysis Runs button, and then select Yield vs.

Process under the Quality Yield buttons. Figure 4.3-12 illustrates a sample yield versus process line graphs for two selected analysis runs.

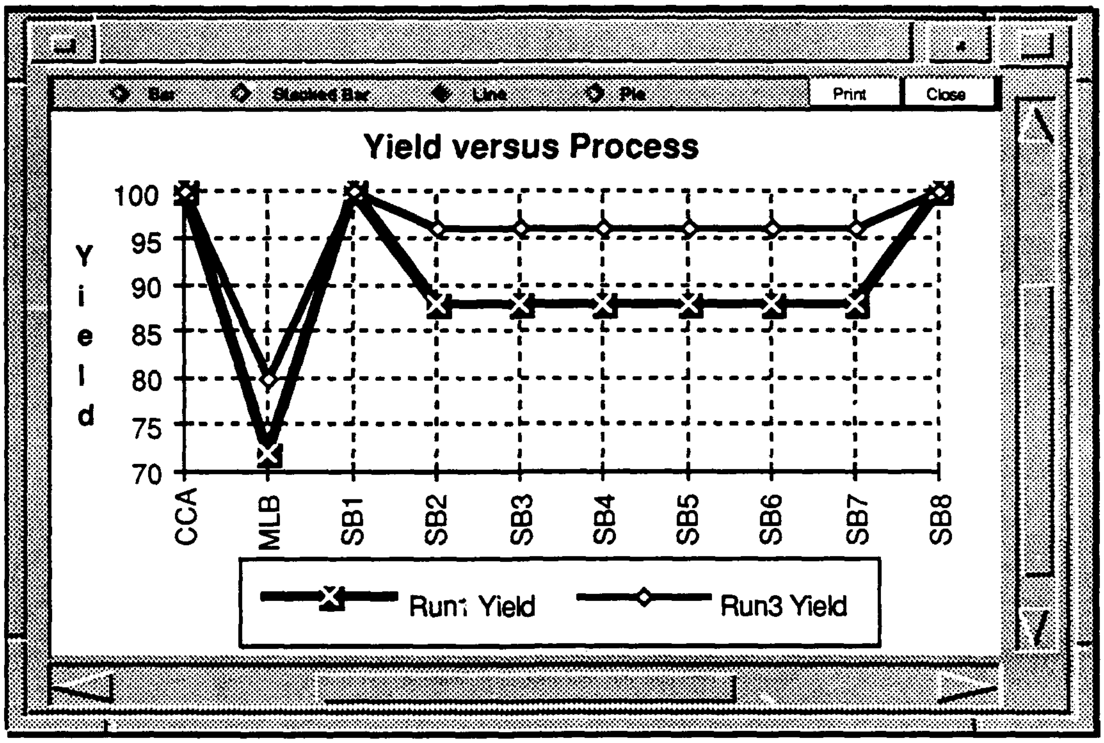


Figure 4.3-12 Yield versus Process Comparison Graph

4.3.3.2 Rework Graphs

The type of rework quality graphs available are rework rates versus processes and rework rates versus operations associated with a particular user selected process. Figure 4.3-13 illustrates the pull out menu for the quality rework graphs. The graphs are also displayed in a separate window, like the yield graphs, where the user can select to display the data as a bar, stacked bar, line, or pie chart. The rework defaulting display will be a bar chart. A sample bar graph of rework versus process is depicted in Figure 4.3-14.

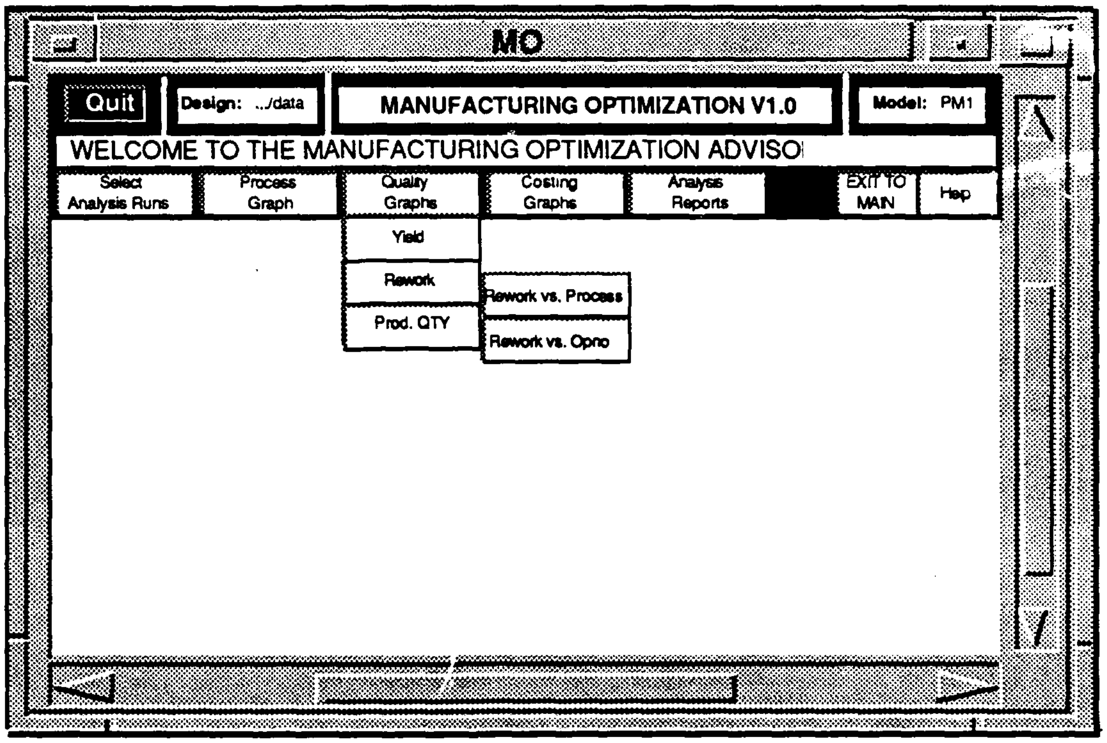


Figure 4.3-13 Rework Graphs

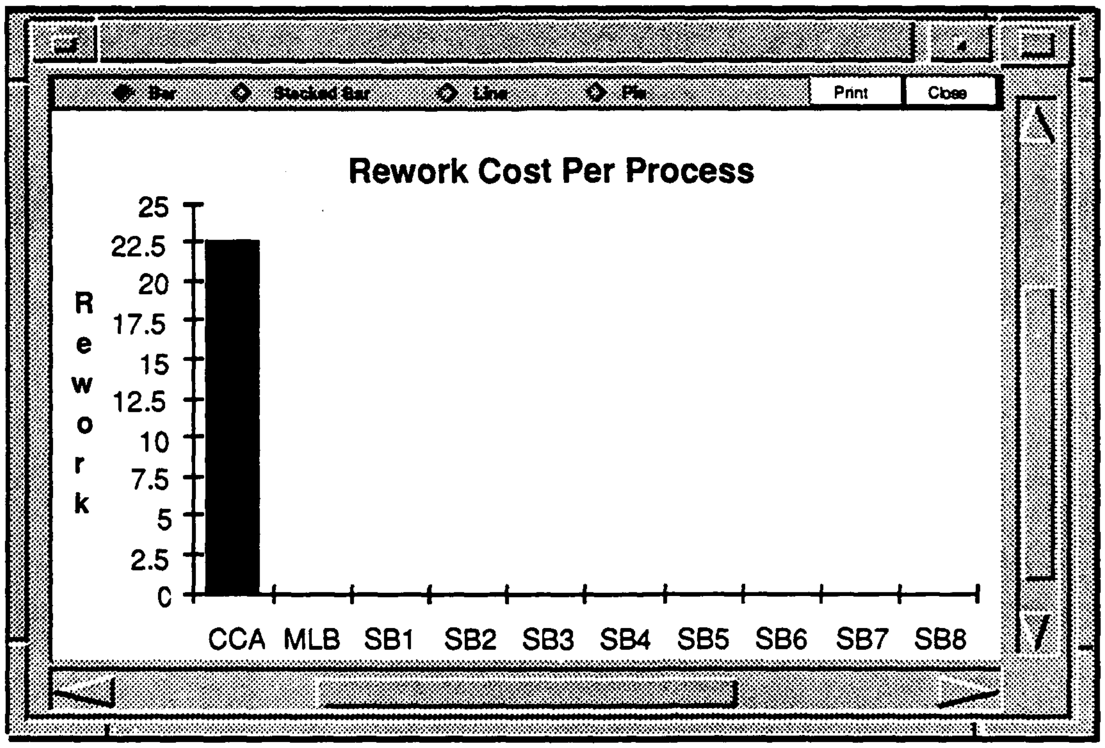


Figure 4.3-14 Rework versus Process Graph

Just like with the Yield versus Process Operations graphing capabilities, the user must select the associated process before a Rework versus Operations graph will be displayed. Figure 4.3-15 illustrates a sample bar chart of rework versus operations.

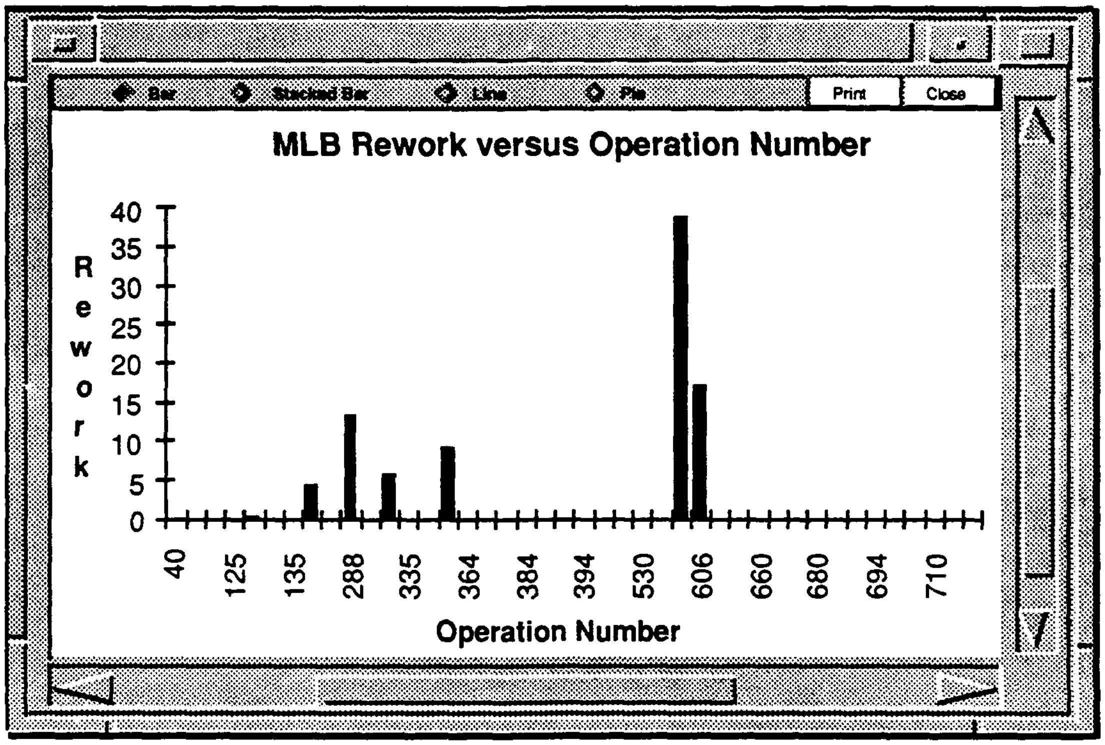


Figure 4.3-15 Rework versus Operations Graph

4.3.3.3 Production Quantity Graphs

The type of production quantity graphs available are quantity rates versus processes and quantity rates versus operations associated with a particular user selected process. Figure 4.3-16 illustrates the pull out menu for the production quantity graphs. The graphs are also displayed in a separate window, like the yield and rework graphs, where the user can select to display the data as a bar, stacked bar, line, or pie chart. The rework defaulting display will be a line chart. A sample line graph of quantity versus process is depicted in Figure 4.3-17.

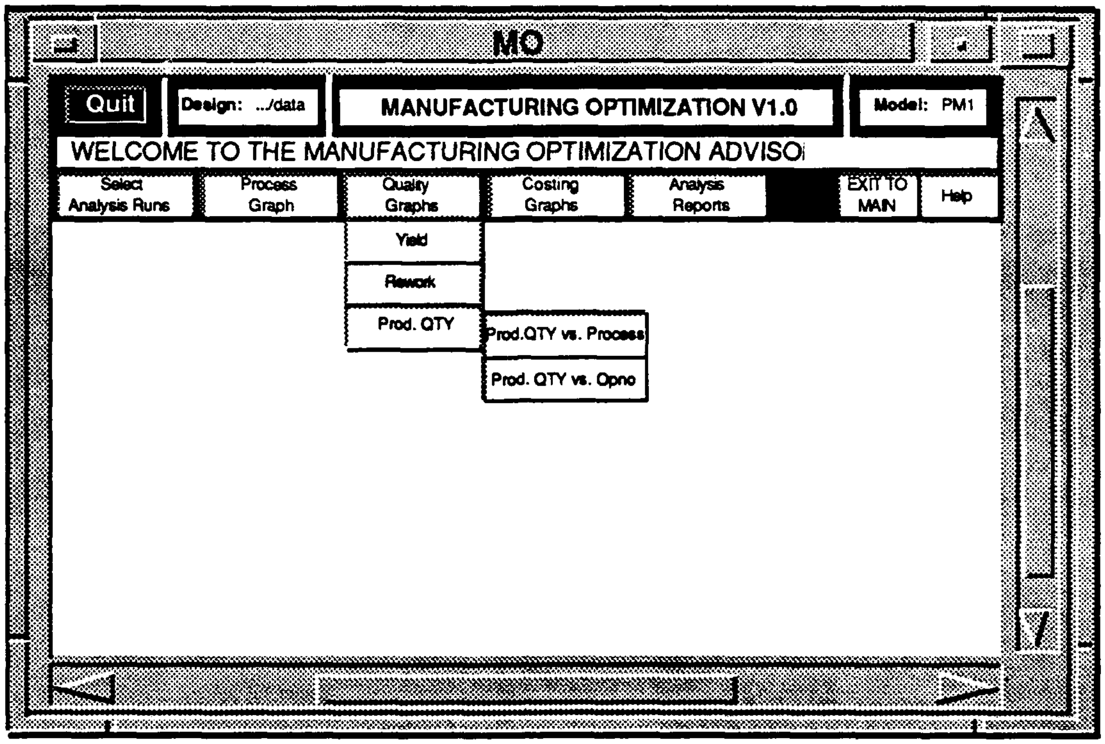


Figure 4.3-16 Production Quantity Graphs

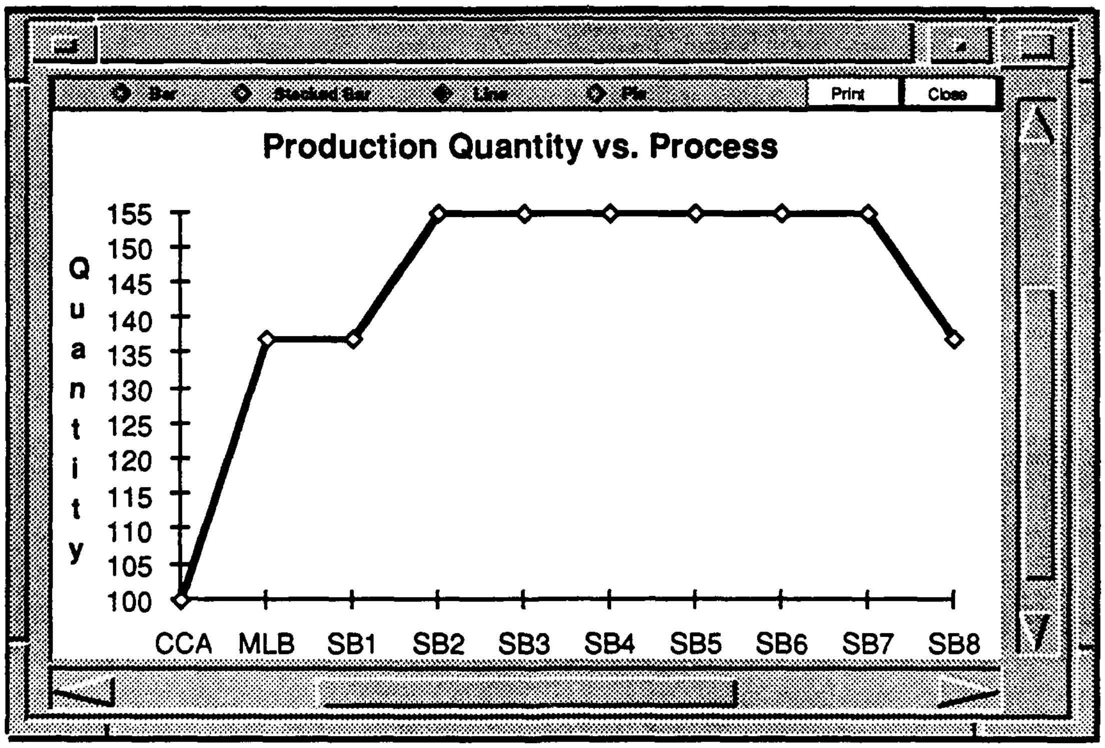


Figure 4.3-17 Production Quantity versus Process Graph

Just like with the Yield and Rework graphing capabilities, the user must select the associated process before a Production Quantity versus Operations graph will be displayed. Figure 4.3-18 illustrates a sample line chart of quantity versus operations.

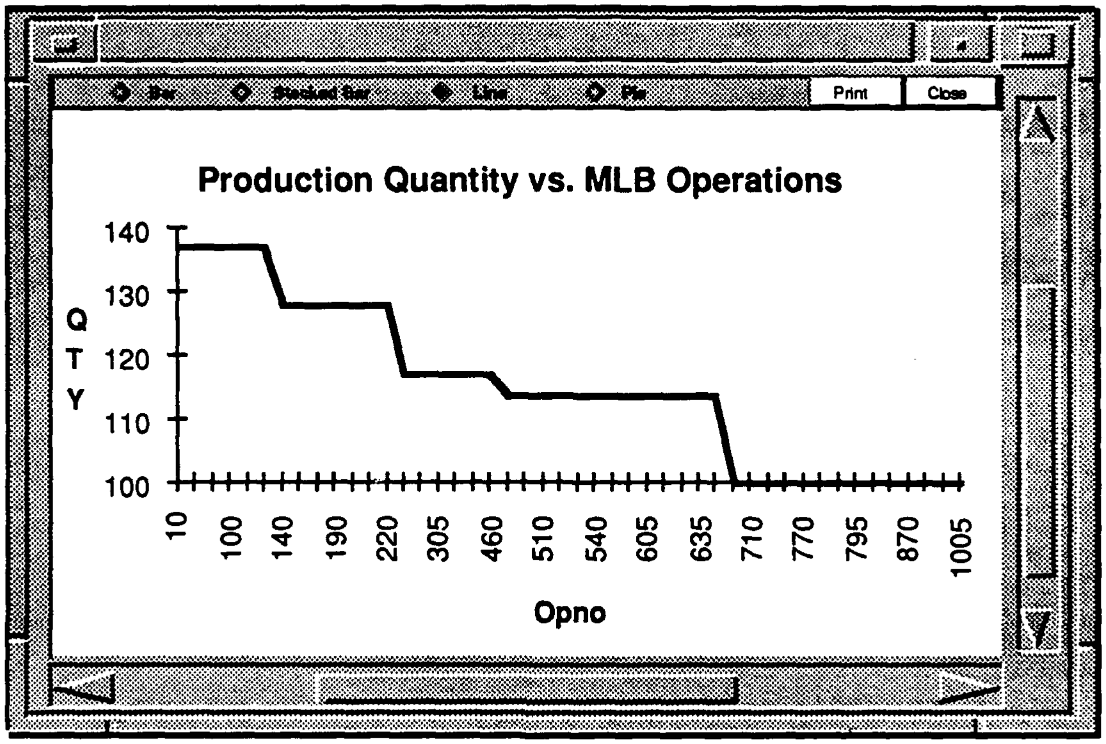


Figure 4.3-18 Production Quantity versus Operations Graph

If the user wants to compare the production quantity rates versus process for three runs, he/she would select the analysis runs from the form under the Select Analysis Runs button, and then select Prod. QTY vs. Process under the Quality Prod. QTY buttons. Figure 4.3-19 illustrates a sample quantity versus process line graphs for three selected analysis runs.

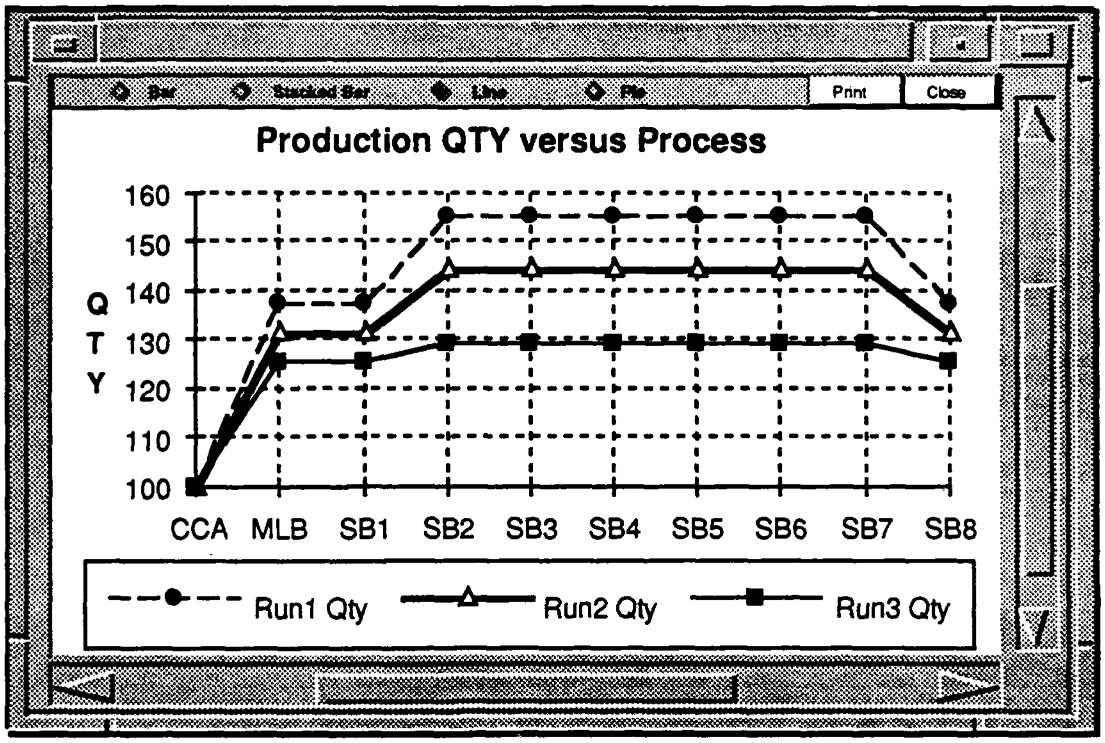


Figure 4.3-19 Production Quantity versus Process Comparison Graph

4.3.4 Available Costing Graphs

The MO system provides for graphically displaying the costing results associated with analysis runs including graphs for time, cost, and cost details. Figure 4.3-20 illustrates the Advisor Window with the Costing Graphs menu pulled down.

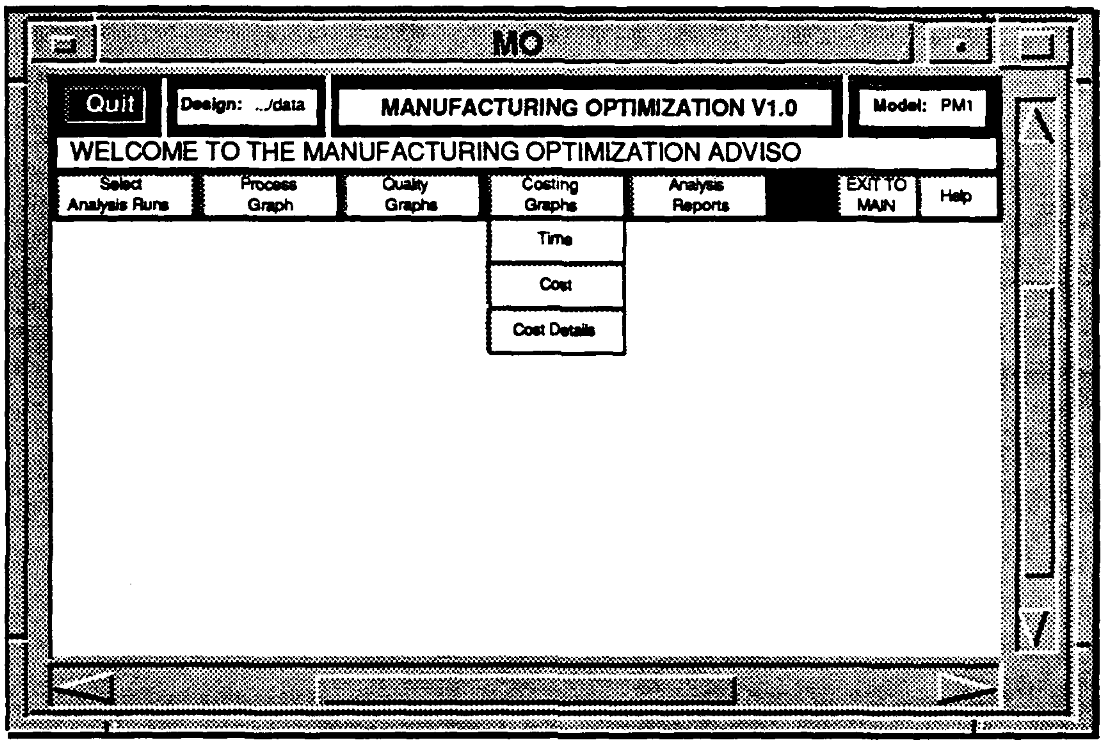


Figure 4.3-20 Costing Graphs

4.3.4.1 Time/Cost Graphs

The type of time/cost quality graphs available are time or cost versus processes and time or cost rates versus operations associated with a particular user selected process. Figure 4.3-21 and figure 4.3-22 illustrate the pull out menu for the quality time and cost graphs, respectively. The graphs are displayed in a separate window where the user can select to display the data as a bar, stacked bar, line, or pie chart. The time default display will be a line chart, and the cost default display will be a bar chart. A sample bar graph of ideal/actual cost versus process is depicted in Figure 4.3-23.

Just like with the Quality graphing capabilities, the user must select the associated process before a Time or Cost versus Operations graph can be displayed. Figure 4.3-24 illustrates a sample bar chart of ideal/actual cost versus operations.

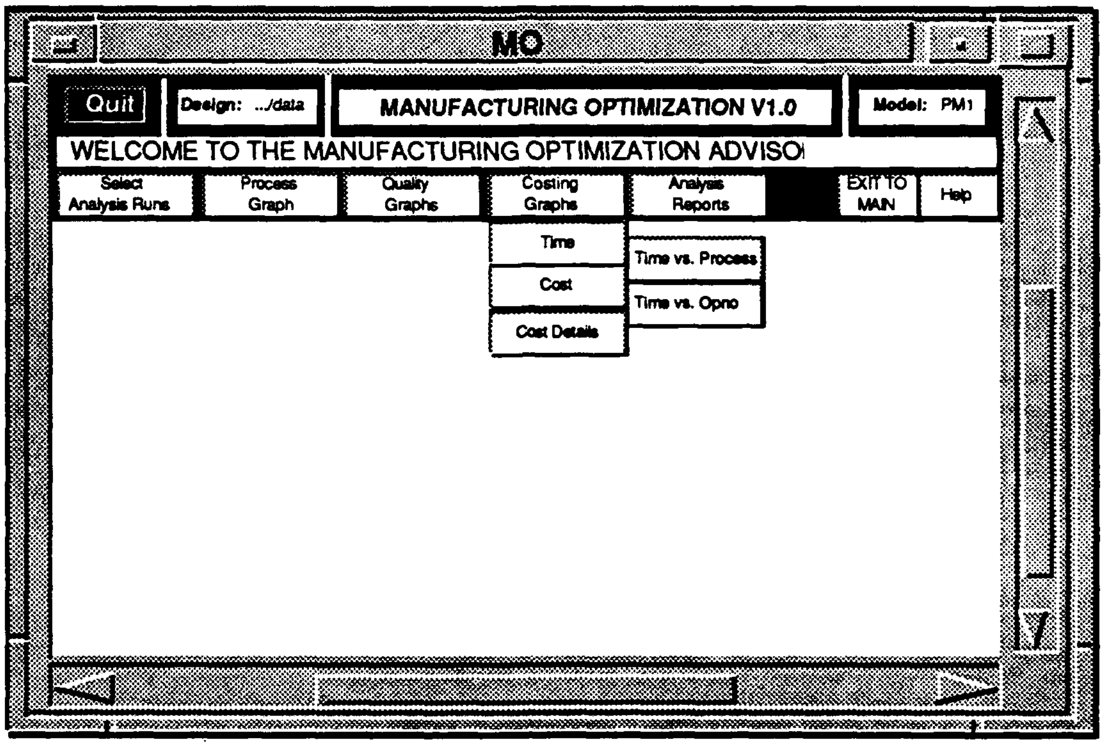


Figure 4.3-21 Labor Time Graphs

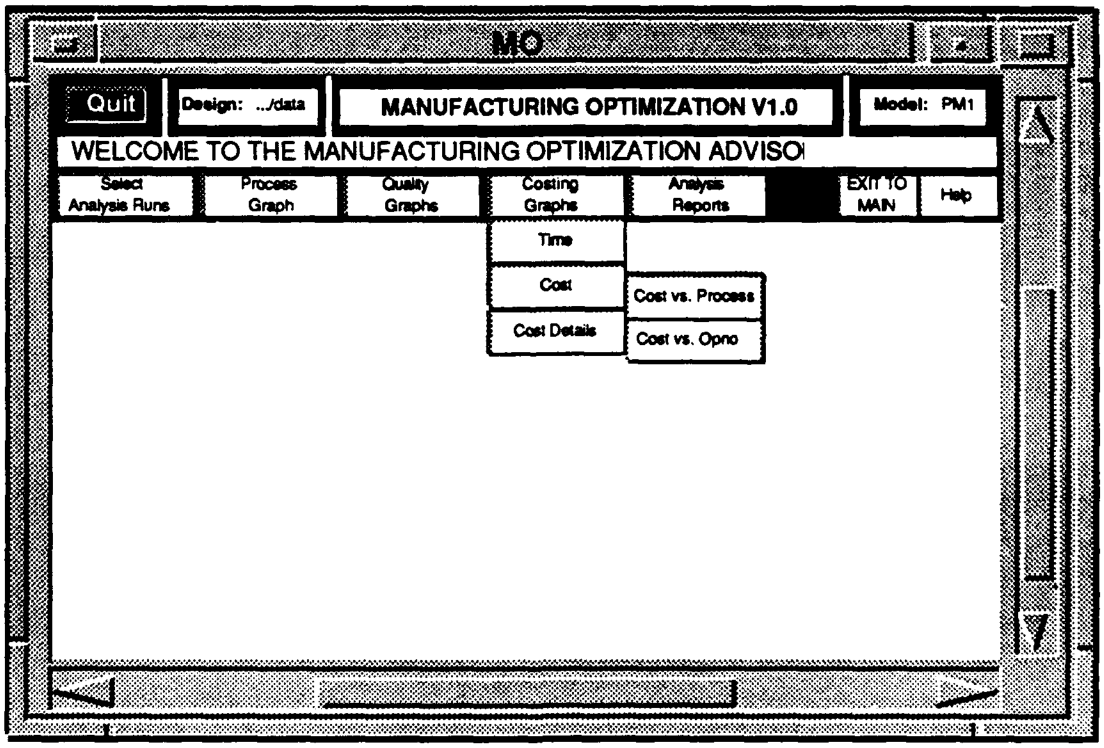


Figure 4.3-22 Cost Graphs

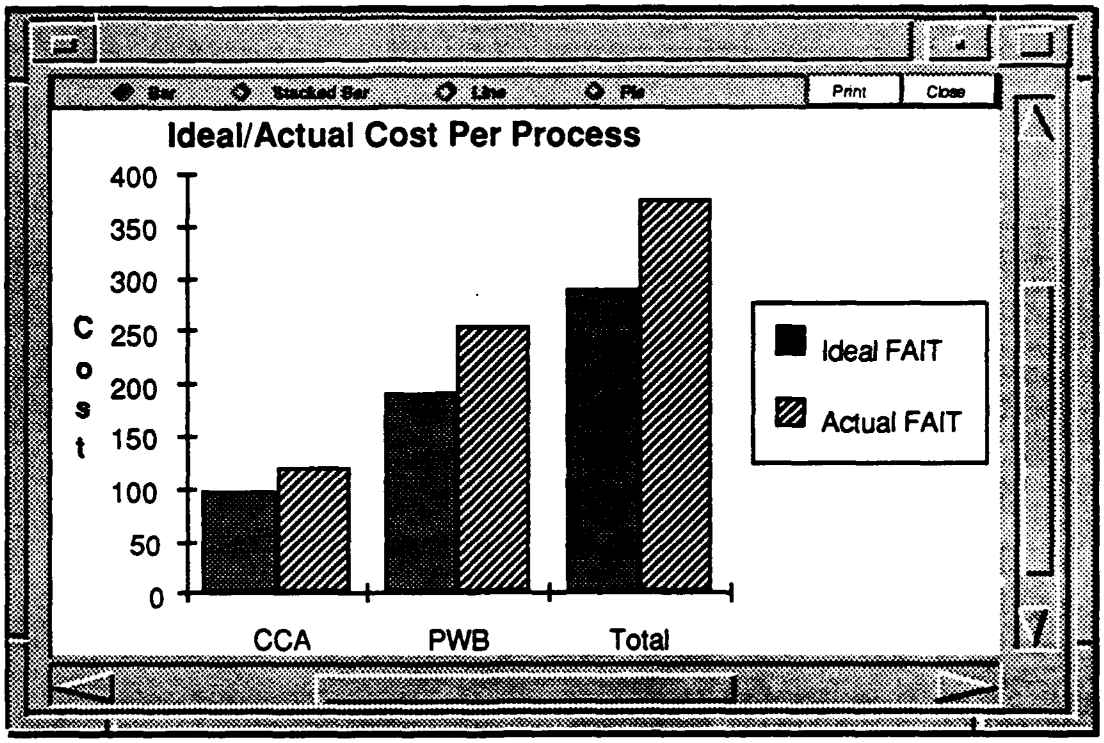


Figure 4.3-23 Cost per Process Graph

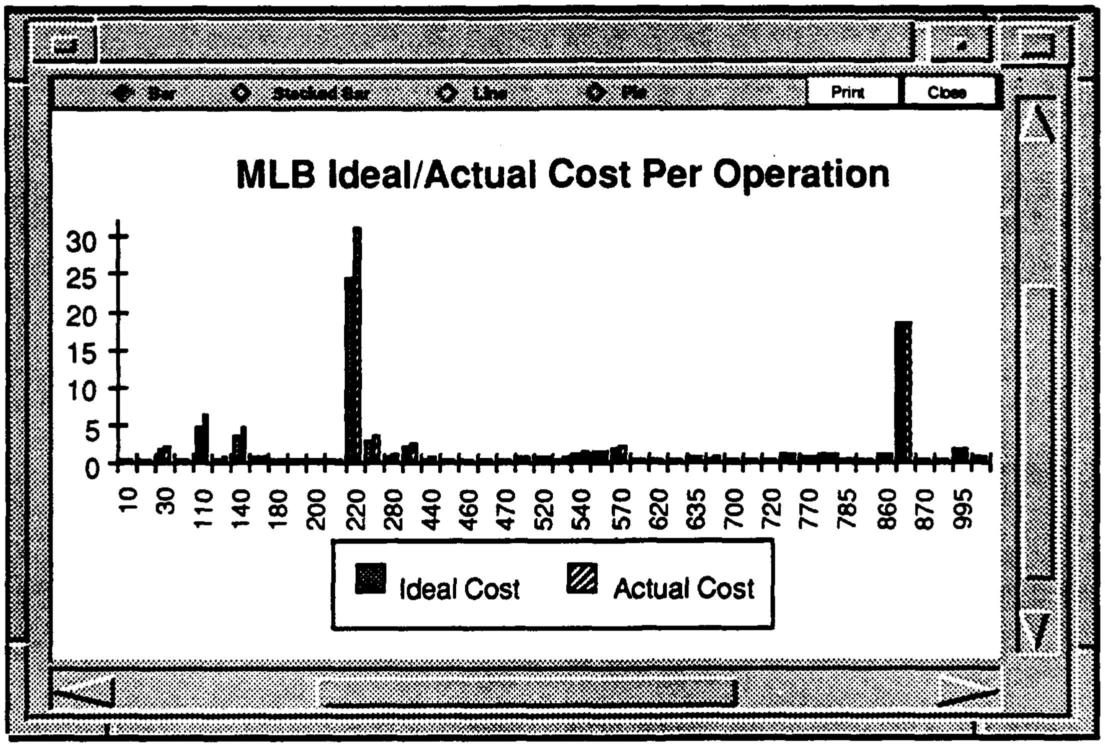


Figure 4.3-24 Cost Per Operation Graph

4.3.4.2 Cost Detail Graphs

The type of cost detail graphs available are product breakdown and process breakdown associated with a particular user selected process(es). The graphs are displayed in a separate window where the user can select to display the data as a bar, stacked bar, line, or pie chart. The cost details default display will be a pie chart. A sample pie chart of a product breakdown is shown in figure 4.3-25, and a sample pie chart of a product process breakdown is shown in figure 4.3-26.

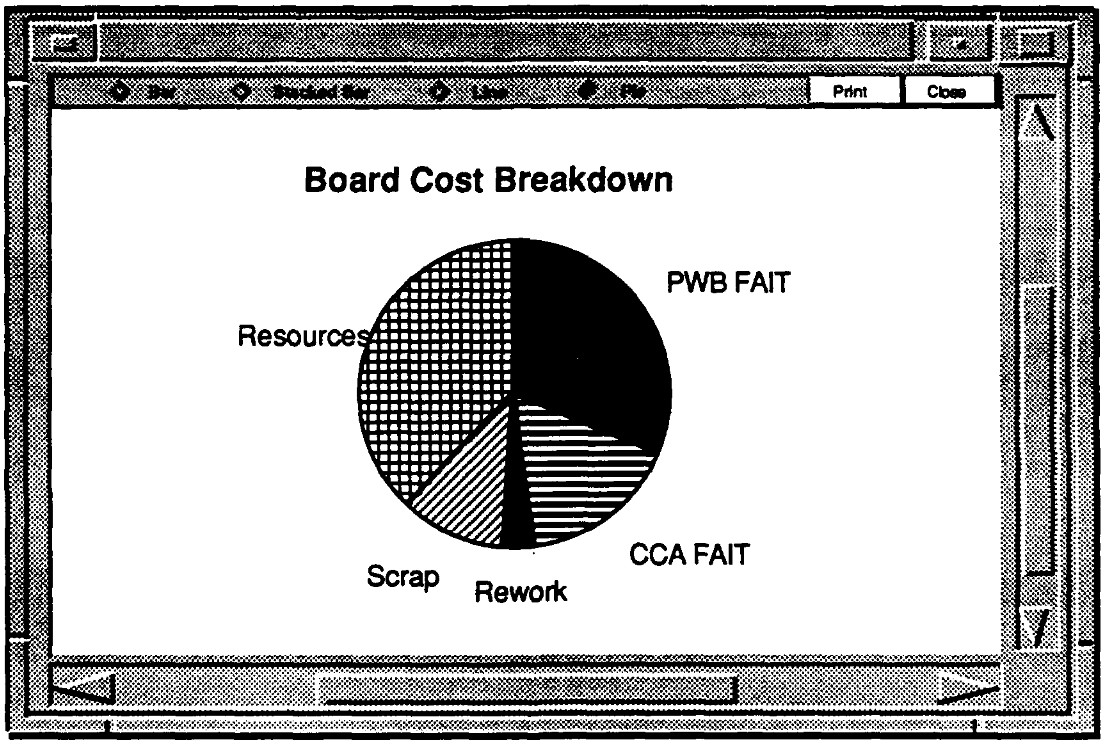


Figure 4.3-25 Cost Details Graph for Product

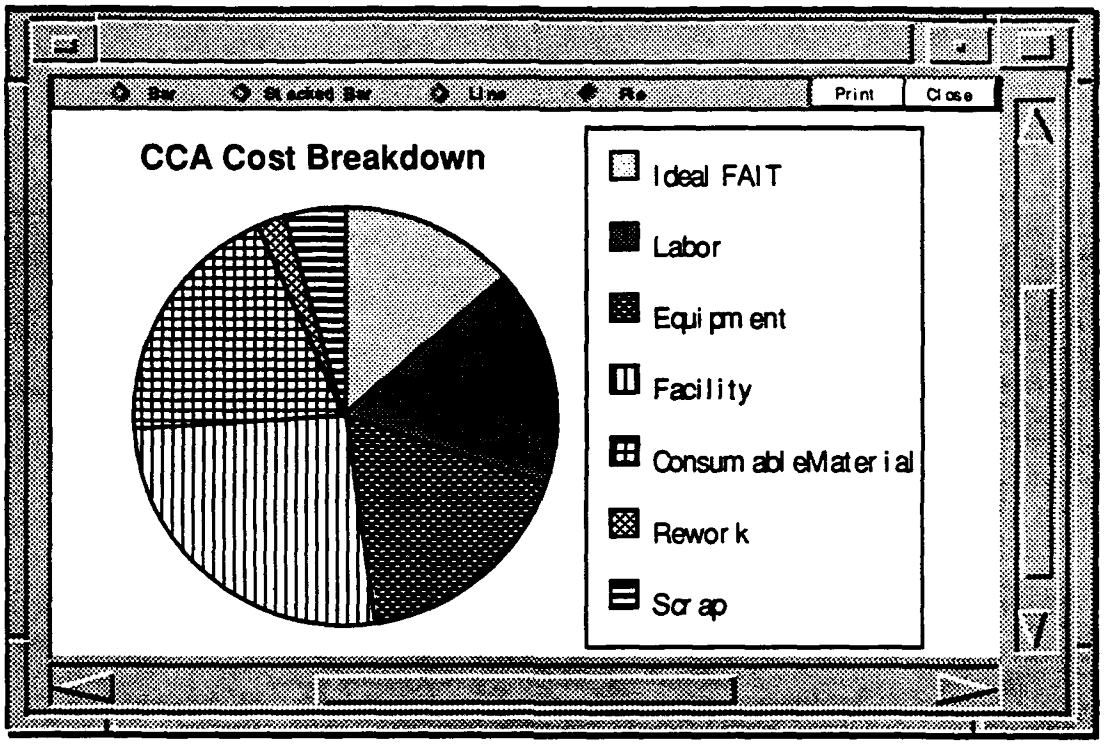


Figure 4.3-26 Cost Details Graph for Process

4.3.5 Analysis Reports Form

The Analysis Report button provides the means to generate reports for the results produced by each process participating in an analysis. This includes the ability to view process flows, yield and rework, cost, and requirements. A final summary report, identifying cost drivers, for each process contributing to a multi-process analysis for a given design database can also be generated. Figure 4.3-27 is displayed when a user selects the analysis report button. The user can then select the type of data that he/she wants in the output report.

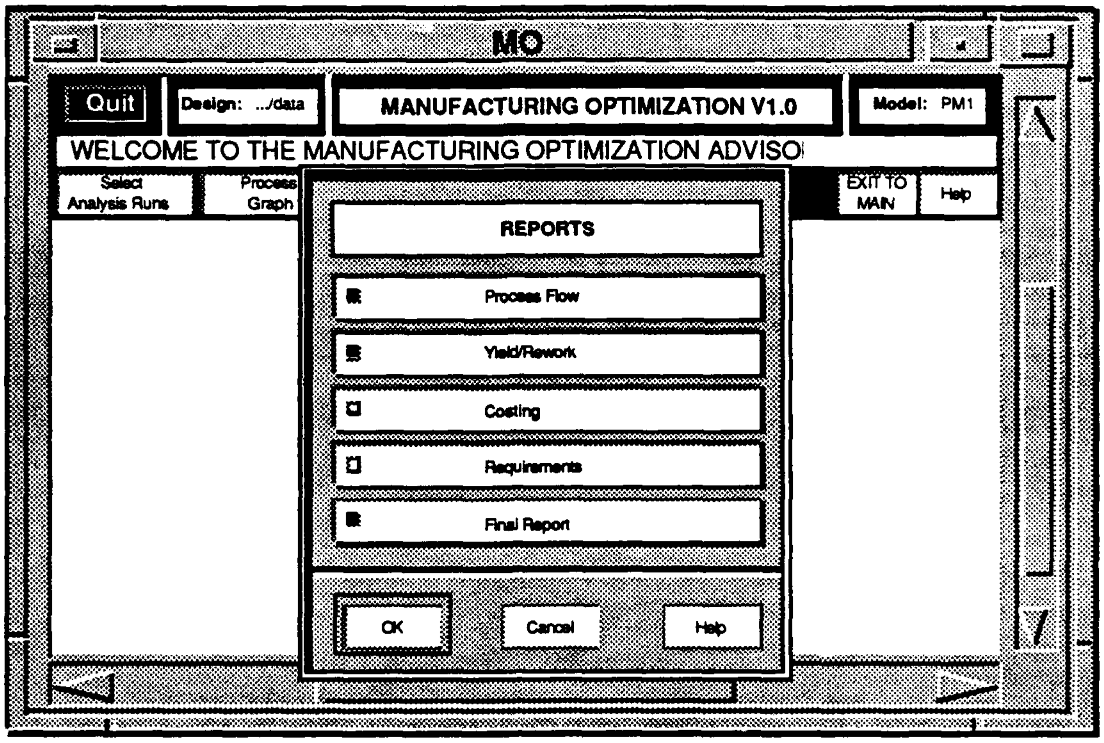


Figure 4.3-27 Analysis Reports Form

Provided below is a sample report generated from the Manufacturing Advisor based on the process flow and corresponding yield results for a PWB Fabrication process.

<u>**Fabrication Process Selection/Cost Estimation Report**</u>

MLB - layers 1, 14 OVERALL YIELD is 94 percent

Opno	*Description*	*Ideal($)*	*Actual($)*	*Rework($)*	*Yield*	*Rework*	*# Units*
10	mark part no	0.123	0.12	0.00	100	0.000	137
30	oxide treat	1.111	1.11	0.00	100	0.000	137
40	bake panels	0.444	0.44	0.00	100	0.000	137
50	lay up	3.123	3.12	0.00	100	0.000	137
60	laminate	0.600	0.80	0.00	94	0.000	137
80	route excess	0.715	0.92	0.00	100	0.000	128
90	oxide strip	0.250	0.32	0.00	100	0.000	128
110	drill tooling	0.220	0.28	0.00	100	0.000	128
130	drill	12.123	15.12	1.23	92	0.005	128
160	electroless	0.661	0.66	0.00	100	0.000	117
170	copper panel	0.555	0.55	0.00	100	0.000	117
180	electrostrike	0.512	0.70	0.00	98	0.000	117

<u>**Fabrication Yield Analysis Report**</u>

MLB - layers 1, 14 OVERALL YIELD IS 94 percent

Opno	*Design Feature Description*	*Value*	*Scrap Per Feature*	*Opno Yield*
60	14 layers and 8 substrates	N/A	6.000	94
130	annular ring	8.00	8.000	92
180	aspect ratio	4.00	2.000	98

4.4 Modeler Window

The Process Modeler will provide manufacturing engineers with the ability to model the manufacturing process of their products. The process model consists of a directed dependency graph. At each vertex of the graph is a process node. Each process node consists of the following:

- Selection rules - If a node's rules are satisfied and its dependencies are satisfied, then the process node is included in the total process analysis model.
- Dependencies (parent process nodes) - For a node to be selected, all of its dependencies must be selected based on the and/or flag.
- Operations - At each process node there is a list of operations that are performed. Each operation is annotated with an associated yield rate, rework rate, and its usage of resources.
- Resources - At each process and operation node there is a list of resources attached. A resource is any facility, person, equipment, or consumable material used in the manufacturing process.

The Process Modeler provides functionality to create new process nodes and edit, copy, and delete existing nodes. Included in this functionality, is a means to specify selection rules for process nodes, define operations that are carried out for a process node, and identify the dependencies among process nodes. Associated with each operation are scrap and rework rates, as well as, resources required to carryout the operation. The Process Modeler window is shown on Figure 4.4-1. A sample process model is displayed.

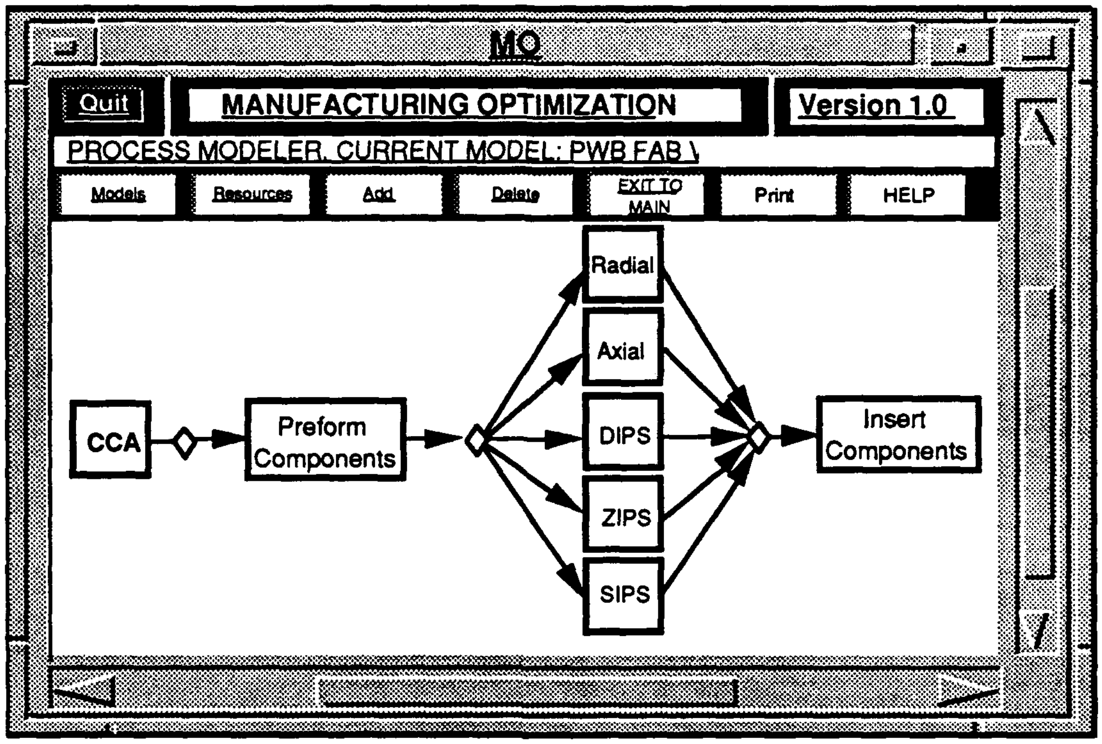


Figure 4.4-1 Process Modeler Window

The user is provided the ability to create new process models and select, delete, and copy existing process models. These operations are done through the Process Model Selection window shown in figure 4.4-2 which is accessed by selecting the Models icon from the Process Model menu bar shown in figure 4.4-1.

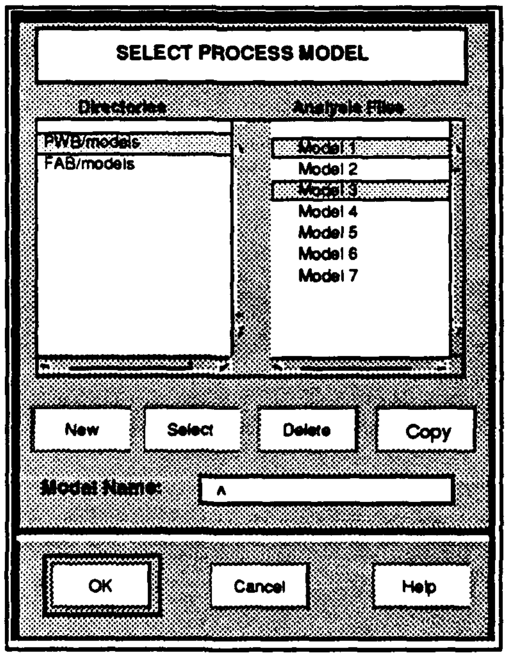


Figure 4.4-2 Process Model Selection Window

4.4.1 Process Node Definition

Defining a new process node will consist of selecting the Add icon from the Process Modeler Window menu bar and specifying the name of the node to the Process Node Specification window shown in figure 4.4-3. The interface will then support the definition of dependencies to other nodes and defining selection rules and operations. It will be possible for process nodes to be copied from a template library or stored to the template library.

Editing existing nodes will be accomplished by graphical selection of the desired node from the process modeler window (see figure 4.4-1) via the mouse. Once the process has been selected, the Process Specification window will be displayed with the data for the selected process node loaded. Existing nodes are deleted by selecting the Delete icon and then the process node to be deleted.

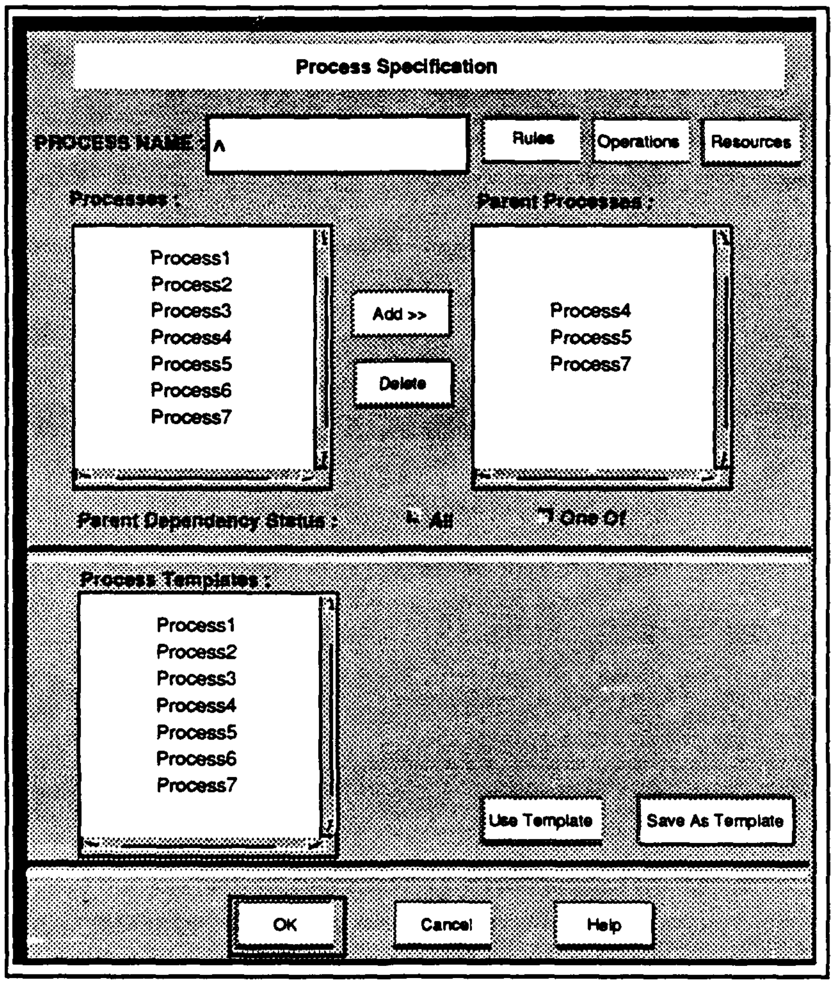
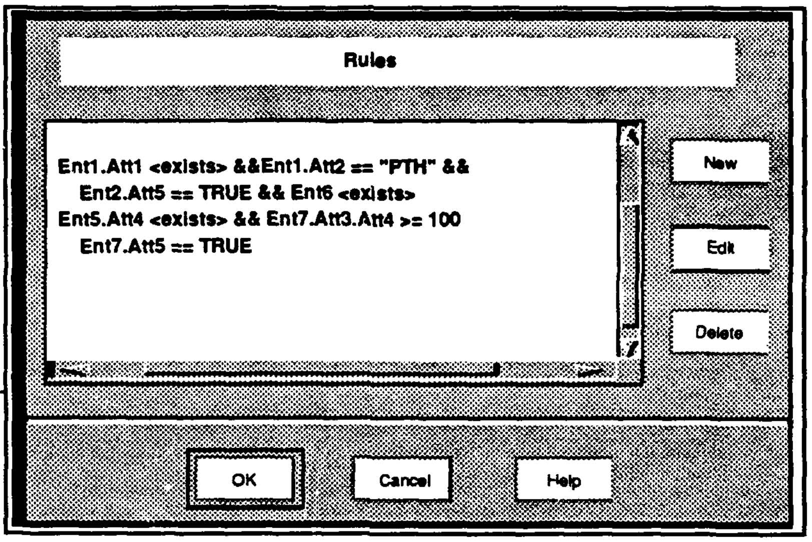


Figure 4.4-3 Process Node Specification Window

4.4.2 Selection Rules Definition

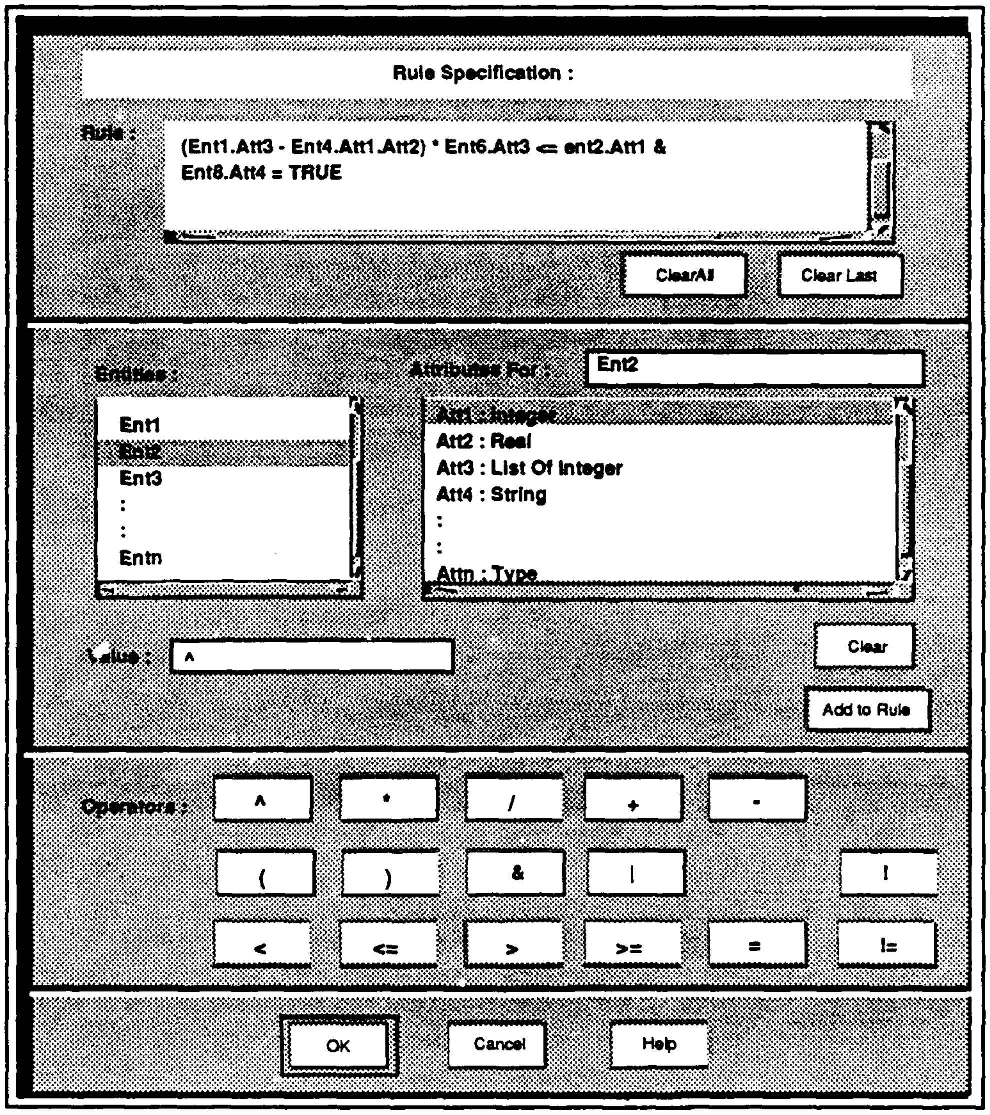
The window in figure 4.4-4 will support the creation, modification, and deletion of selection rules for a process node. There will be an implicit OR between each of the rules in the list for a process node, i.e., if one of the rules in the list is satisfied, then the node will be selected.



```
Ent1.Att1 <exists> &&Ent1.Att2 == "PTH" &&
   Ent2.Att5 == TRUE && Ent6 <exists>
Ent5.Att4 <exists> && Ent7.Att3.Att4 >= 100
   Ent7.Att5 == TRUE
```

Figure 4.4-4 Selection Rules Window

When either a new rule is to be defined or an existing one modified, the Rule Specification window shown in figure 4.4-5 will be presented. This window will also be utilized to specify scrap and rework rate rules and operation setup and operation run time rules.



Rule Specification :

Rule : (Ent1.Att3 - Ent4.Att1.Att2) * Ent6.Att3 <= ent2.Att1 & Ent8.Att4 = TRUE

Entities : Ent1, Ent2, Ent3, :, :, Entn

Attributes For : Ent2

Att1 : Integer
Att2 : Real
Att3 : List Of Integer
Att4 : String
:
:
Attn : Type

Value : ^

Figure 4.4-5 Rule Specification

4.4.3 Operation Definition

The window in figure 4.4-6 will support the creation, modification, and deletion of operations for a process node.

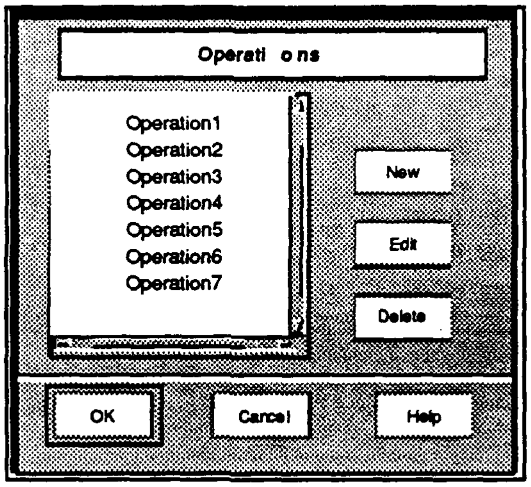


Figure 4.4-6 Operations Window

Operations will be created and modified through the Operations Specification window shown in figure 4.4-7. Operations can be stored in and copied from a library of existing operations. Attached to each operation will be a list of resources that are utilized to perform the operation. Also attached to operations will be a list of scrap and rework rate rules.

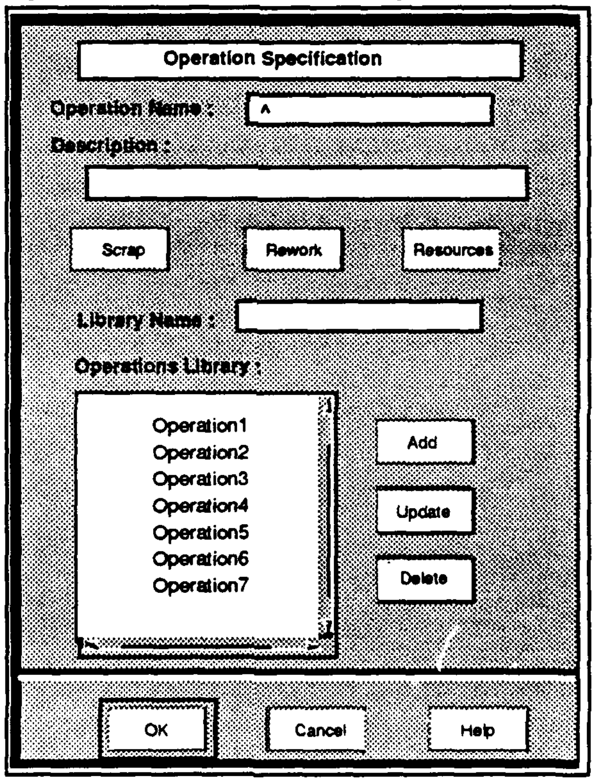
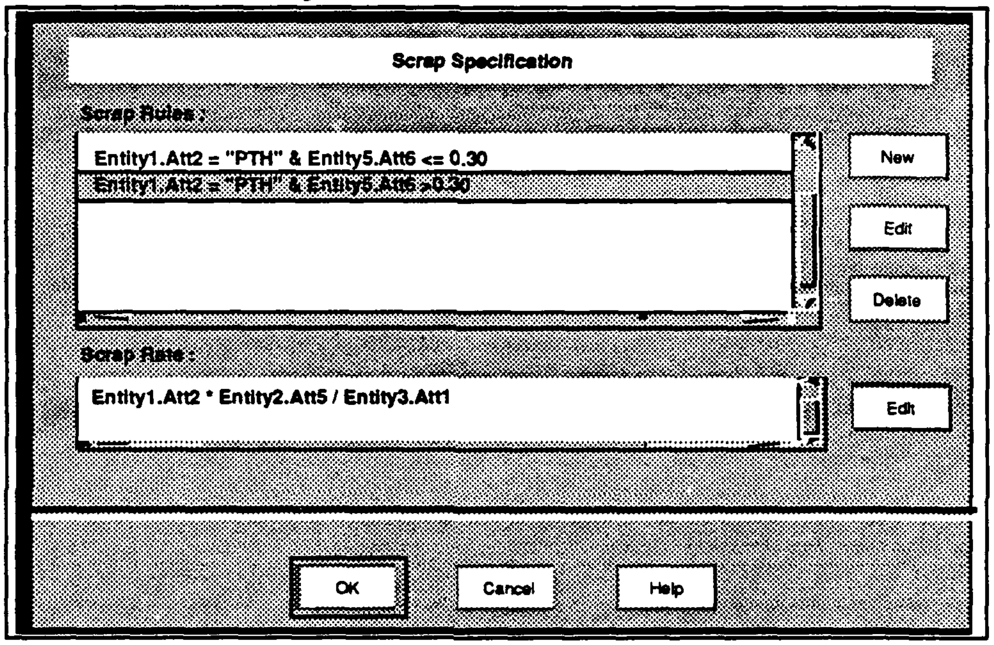


Figure 4.4-7 Operations Specification Window

4.4.3.1 Scrap Rate Definition

Attached to every operation is a list of scrap rates. A scrap rate can be associated for a given entity attribute or a set of entity attributes. This will be specified through an ordered list of rules and scrap rates. The scrap rate will be established using the scrap rate equation attached to the first scrap rule in the ordered list that is satisfied. A user interface supporting this functionality is shown in figure 4.4-8. The scrap rules and the scrap rates will be specified through the Rule Specification Window shown in figure 4.4-5.



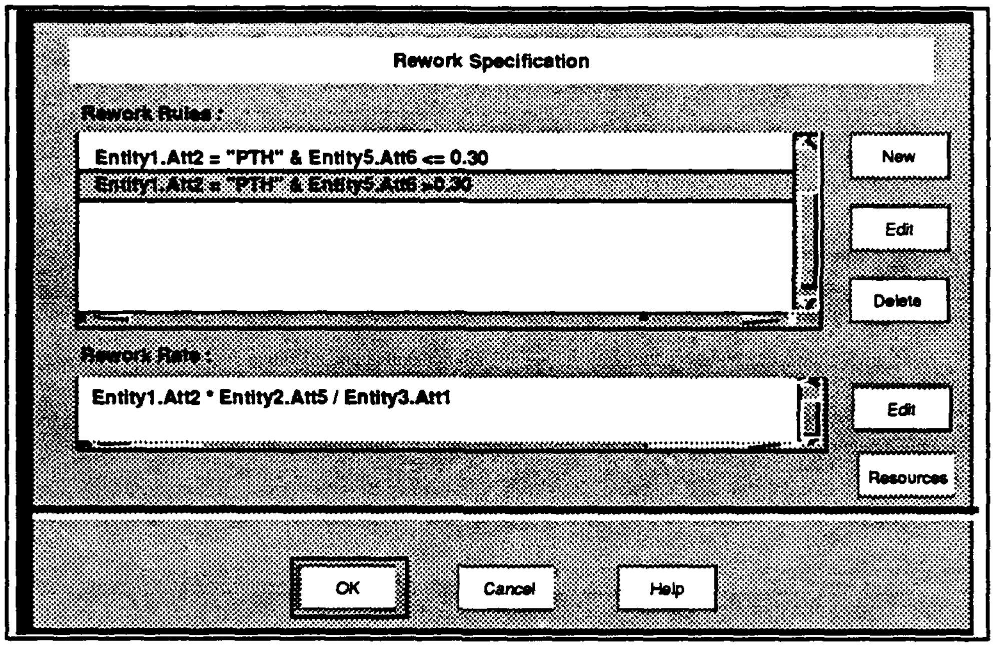
Entity1.Att2 = "PTH" & Entity5.Att6 <= 0.30
Entity1.Att2 = "PTH" & Entity5.Att6 >0.30

Entity1.Att2 * Entity2.Att5 / Entity3.Att1

Figure 4.4-8 Scrap Specification Window

4.4.3.2 Rework Rate Definition

Attached to every operation is a list of rework rates. A rework rate can be associated for a given entity attribute or a set of entity attributes. This will be specified through an ordered list of rules and rework rates. The rework rate will be established using the rework rate equation attached to the first rework rule in the ordered list that is satisfied. A user interface supporting this functionality is shown in figure 4.4-9. The rework rules and the rework rates will be specified through the Rule Specification Window shown in figure 4.4-5. Also attached to each rework rule rate pair is a list of resources that is required to complete the rework activities. The resources used along with their associated setup and run time equations will specified using the Resource Utilization interface shown in figure 4.4-10.



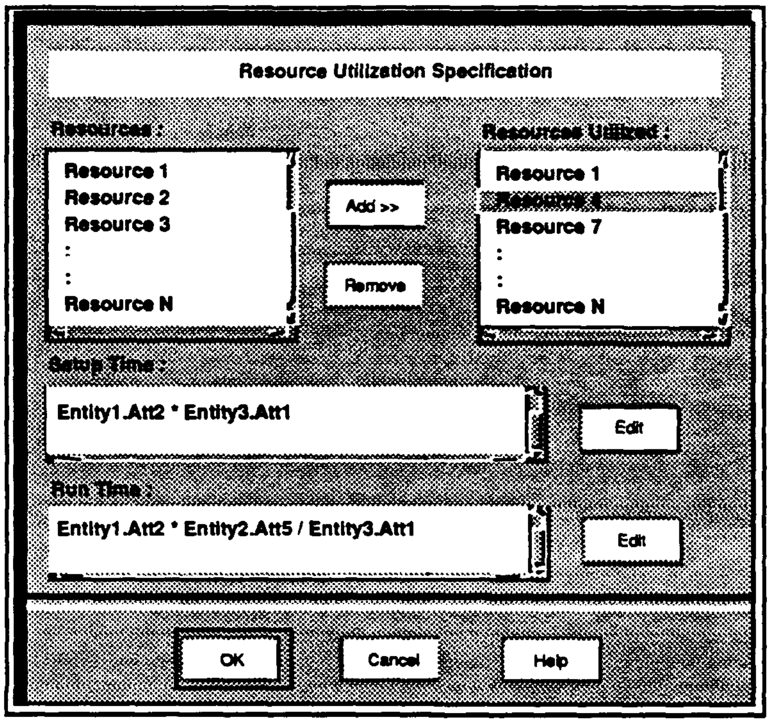
Entity1.Att2 = "PTH" & Entity5.Att6 <= 0.30
Entity1.Att2 = "PTH" & Entity5.Att6 > 0.30

Entity1.Att2 * Entity2.Att5 / Entity3.Att1

Figure 4.4-9 Rework Specification Window

4.4.4 Resource Definition

For each operation performed, a list of needed resources must be specified. When resources are utilized the amount of setup time and run time that is required for the resource must also be provided so that proper costing can be calculated. Figure 4.4-10 shows the Resource Utilization interface that will allow the process modeler to construct the list of resources utilized by an operation. The setup and run time equations are specified using the Rule Specification interface shown in figure 4.4-5.



Resource Utilization Specification

Resources :

Resource 1
Resource 2
Resource 3
:
:
Resource N

Resources Utilized :

Resource 1
Resource 4
Resource 7
:
:
Resource N

Setup Time :

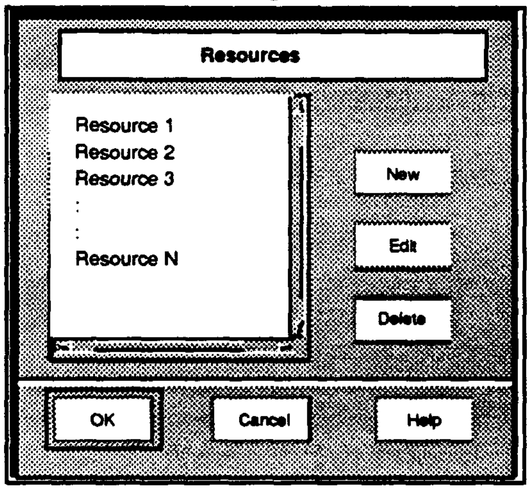
Entity1.Att2 * Entity3.Att1

Run Time :

Entity1.Att2 * Entity2.Att5 / Entity3.Att1

Figure 4.4-10 Resource Utilization Window

Figure 4.4-11 show the Resources interface which lists all of the Resources that are currently stored in the process model. The interface supports creating new resources, and editing and deleting existing resources. To access the Resources interface the user would select the Resources icon from the menu bar shown in figure 4.4-1.

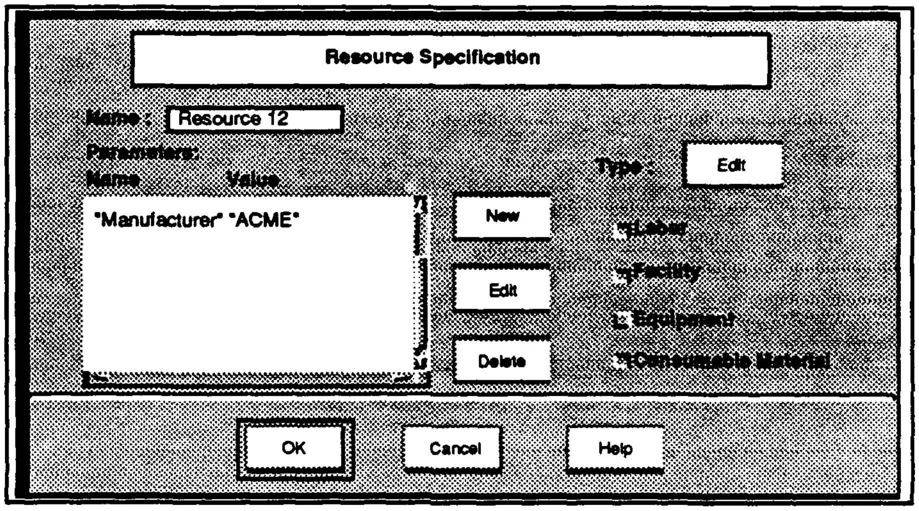


Resources

Resource 1
Resource 2
Resource 3
:
:
Resource N

Figure 4.4-11 Resource Window

The Resource Specification interface is shown in figure 4.4-12. This interface is used for specifying new resources and modifying existing ones. Attached to each resource is a list of user definable parameters or attributes. Each resource falls into one of the following four categories: labor, facility, equipment, or consumable resource.



Name : Resource 12

"Manufacturer" "ACME"

Figure 4.4-12 Resource Specification Window

Figure 4.4-13 shows the interface for specifying user definable parameters for resources.

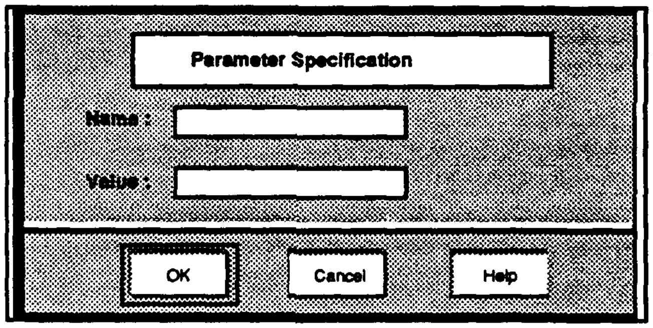


Figure 4.4-13 Resource Parameter Specification Window

Figure 4.4-14 shows the interface for specifying a facility resource.

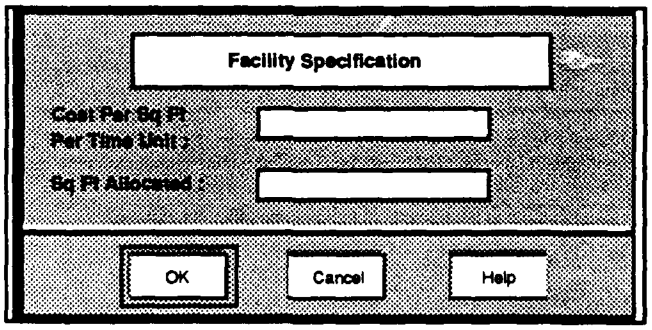
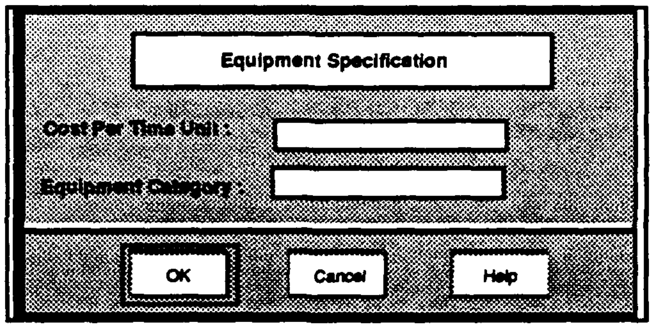


Figure 4.4-14 Facility Resource Specification Window

Figure 4.4-15 shows the interface for specifying an equipment resource.

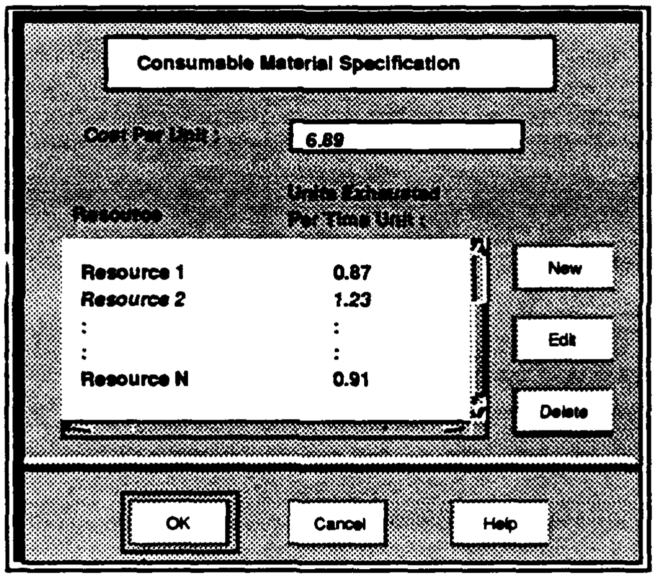


Cost Per Time Unit :

Equipment Category :

Figure 4.4-15 Equipment Resource Specification Window

Figure 4.4-16 shows the interface for specifying a consumable material resource. Materials are consumed at different rates by different resources therefore a list of resources with the rate at which the material is consumed is provided.

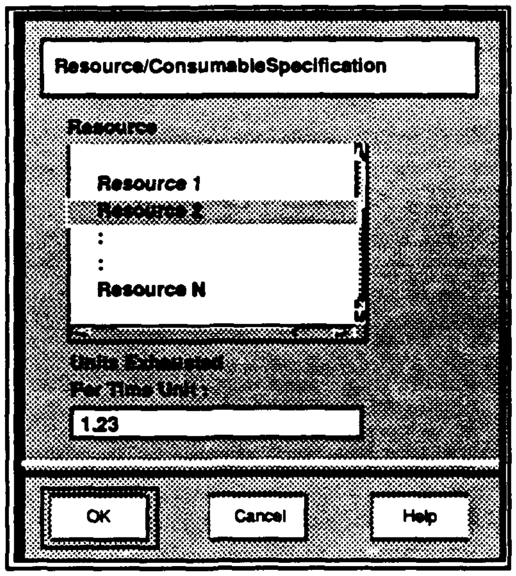


Cost Per Unit : 6.89

Resource	Units Exhausted Per Time Unit :
Resource 1	0.87
Resource 2	*1.23*
:	:
:	:
Resource N	0.91

Figure 4.4-16 Consumable Material Resource Specification Window

An individual resource, consumable material pair is specified in the Resource/Consumable Specification Window shown in figure 4.4-17.



Resource/ConsumableSpecification

Resource

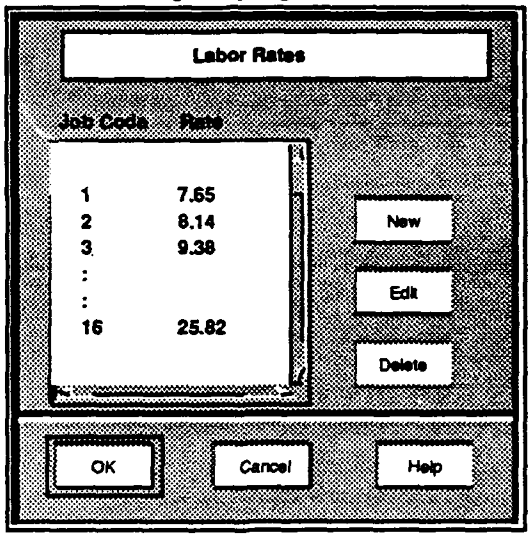
Resource 1
Resource 2
:
:
Resource N

Units Exhausted Per Time Unit:

1.23

Figure 4.4-17 Resource/Consumable Specification Window

Figure 4.4-18 shows the interface for specifying labor resources.

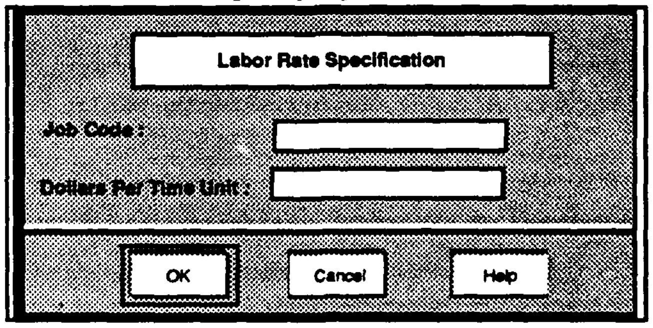


Labor Rates

Job Code	Rate
1	7.65
2	8.14
3	9.38
:	
:	
16	25.82

Figure 4.4-18 Labor Resource Window

Figure 4.4-19 shows the interface for specifying a labor rate resource.



Labor Rate Specification

Job Code :

Dollars Per Time Unit :

Figure 4.4-19 Labor Rate Resource Specification Window

5. Detailed Object Oriented Design

This section provides an object oriented specification of the MO system. It contains the detail design of the underlying classes and objects which make up the system, excluding the user interface. The detailed design of the user interface can be found in section 4, where we have prototyped the look and feel of the system. The Object Oriented Design (OOD) methodology used is defined in Object Oriented Design with Applications by Grady Booch, (See reference 7 in section 2). The Booch OOD methodology provides a means of graphically illustrating class and object hierarchies and relationships, definition of system state diagrams, and the specification of class methods in the form of C++ code directly.

The class specifications defined in this section were developed as follows: the EXPRESS information modeling language was used to model both the product data and the MO process data (Section 6 provides a complete EXPRESS schema specification of the product and process models). Using the express2c++ compiler which is part of the STEP Programmers Toolkit (STEP Tools Inc.), the EXPRESS entities were translated into C++ classes. The generated classes are structured such that all of the class attributes are declared as private. Public access and update methods were generated for each private attribute. Additional calculation and monitoring methods required for each class were then added to the class manually.

Figure 5-1 illustrates the top-level class categories for the MO system. The sections to follow will provide the details of each of these categories, including the applicable class and object diagrams, state transition diagrams, and class specifications.

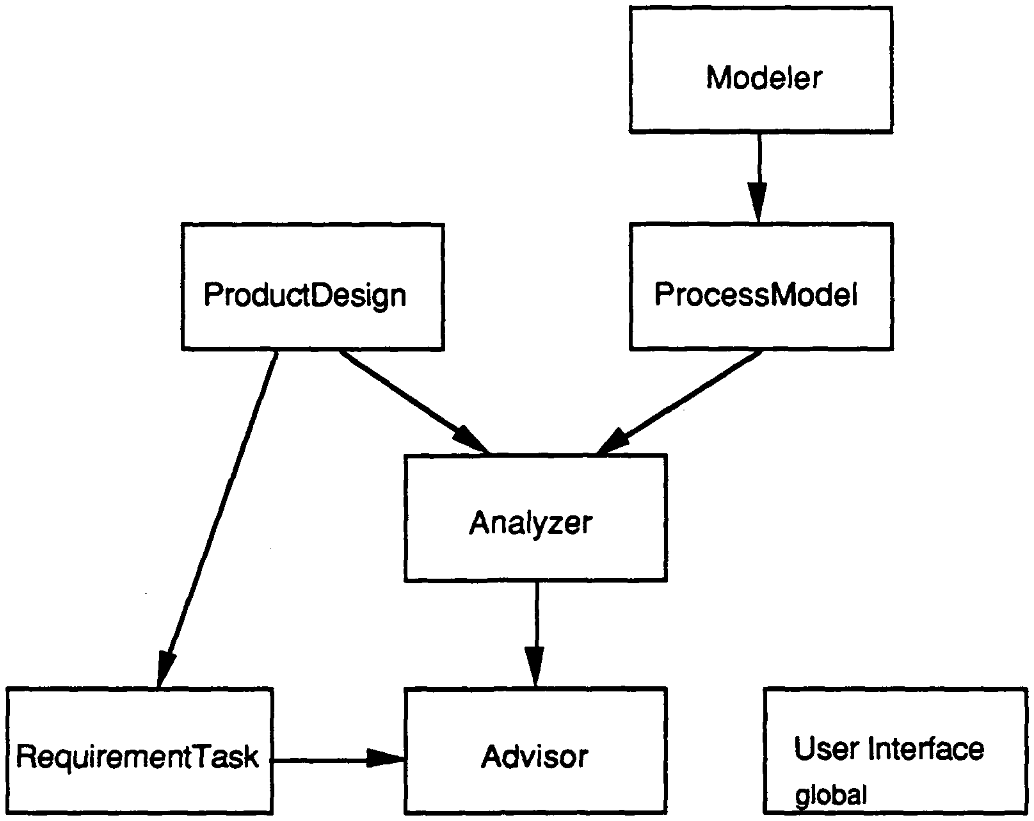


Figure 5-1 Top-Level Class (Categories) Diagram

5.1 ProductDesign

An EXPRESS product model was developed to model PWB data and electronic component library data (See section 6.2 for this specification). The model consists of approximately twenty interrelated EXPRESS schemas consisting of more than one hundred and fifty entities. C++ source code was produced by the express2c++ compiler as described above. The following specification is for the "route_rec" C++ class which corresponds to the "route_rec" EXPRESS entity defined in section 6.2.1.15.

```
/* Class Declaration */
ROSE_DECLARE (route_rec) : virtual public RoseStructure {
 private:
        STR PERSISTENT_signal;
        STR PERSISTENT_route_type;
        STR PERSISTENT_status;
        pin_name_rec * PERSISTENT_target_name;
        pin_name_rec * PERSISTENT_object_name;
        pin_rec * PERSISTENT_target_pin;
        pin_rec * PERSISTENT_object_pin;
        point_rec * PERSISTENT_target_loc;
        point_rec * PERSISTENT_object_loc;
        BOOL PERSISTENT_protect;
        int PERSISTENT_target_layer;
        int PERSISTENT_object_layer;
        ListOfsegment_rec * PERSISTENT_path;
        int PERSISTENT_shield_id;
        int PERSISTENT_pin_pair_index;
        pin_pair_rec * PERSISTENT_pin_pair;
        ww_route_data_rec * PERSISTENT_ww_data; STR PERSISTENT_comment;
 public:
        ROSE_DECLARE_MEMBERS(route_rec);

/* Access and Update Methods */
/* signal Access Methods */
STR signal()
{       return ROSE_GET_PRIM (STR,PERSISTENT_signal);
}

void signal (STR asignal)
{       ROSE_PUT_PRIM (STR,PERSISTENT_signal,asignal); }

/* route_type Access Methods */
STR route_type()
{       return ROSE_GET_PRIM (STR,PERSISTENT_route_type);
}

void route_type (STR aroute_type)
{       ROSE_PUT_PRIM (STR,PERSISTENT_route_type,aroute_type); }

/* status Access Methods */
STR status()
{       return ROSE_GET_PRIM (STR,PERSISTENT_status);
}

void status (STR astatus)
{       ROSE_PUT_PRIM (STR,PERSISTENT_status,astatus); }

/* target_name Access Methods */
pin_name_rec * target_name()
{       return ROSE_GET_OBJ (pin_name_rec,PERSISTENT_target_name);
}

void target_name (pin_name_rec * atarget_name)
{       ROSE_PUT_OBJ (pin_name_rec,PERSISTENT_target_name,atarget_name); }

/* object_name Access Methods */
pin_name_rec * object_name()
{       return ROSE_GET_OBJ (pin_name_rec,PERSISTENT_object_name);
}

void object_name (pin_name_rec * aobject_name)
{       ROSE_PUT_OBJ (pin_name_rec,PERSISTENT_object_name,aobject_name); }

/* target_pin Access Methods */
pin_rec * target_pin()
```

```
{       return ROSE_GET_OBJ (pin_rec,PERSISTENT_target_pin);
}

void target_pin (pin_rec * atarget_pin)
{       ROSE_PUT_OBJ (pin_rec,PERSISTENT_target_pin,atarget_pin); }

/* object_pin Access Methods */
pin_rec * object_pin()
{       return ROSE_GET_OBJ (pin_rec,PERSISTENT_object_pin);
}

void object_pin (pin_rec * aobject_pin)
{       ROSE_PUT_OBJ (pin_rec,PERSISTENT_object_pin,aobject_pin); }

/* target_loc Access Methods */
point_rec * target_loc()
{       return ROSE_GET_OBJ (point_rec,PERSISTENT_target_loc);
}

void target_loc (point_rec * atarget_loc)
{       ROSE_PUT_OBJ (point_rec,PERSISTENT_target_loc,atarget_loc); }

/* object_loc Access Methods */
point_rec * object_loc()
{       return ROSE_GET_OBJ (point_rec,PERSISTENT_object_loc);
}

void object_loc (point_rec * aobject_loc)
{       ROSE_PUT_OBJ (point_rec,PERSISTENT_object_loc,aobject_loc); }

/* protect Access Methods */
BOOL protect()
{       return ROSE_GET_PRIM (BOOL,PERSISTENT_protect);
}

void protect (BOOL aprotect)
{       ROSE_PUT_PRIM (BOOL,PERSISTENT_protect,aprotect); }

/* target_layer Access Methods */
int target_layer()
{       return ROSE_GET_PRIM (int,PERSISTENT_target_layer);
}

void target_layer (int atarget_layer)
{       ROSE_PUT_PRIM (int,PERSISTENT_target_layer,atarget_layer); }

/* object_layer Access Methods */
int object_layer()
{       return ROSE_GET_PRIM (int,PERSISTENT_object_layer);
}

void object_layer (int aobject_layer)
{       ROSE_PUT_PRIM (int,PERSISTENT_object_layer,aobject_layer`, }
/* path Access Methods */
ListOfsegment_rec * path();
void path (ListOfsegment_rec *  apath)
{       ROSE_PUT_OBJ (ListOfsegment_rec,PERSISTENT_path,apath); }

ListOfsegment_rec * route_rec :: path()
{       if( !PERSISTENT_path)
                if( this->isPersistent())
                        path (pnewIn (design()) ListOfsegment_rec);
        else    path (new ListOfsegment_rec);
        return ROSE_GET_OBJ (ListOfsegment_rec,PERSISTENT_path);
}


/* shield_id Access Methods */
int shield_id()
{       return ROSE_GET_PRIM (int,PERSISTENT_shield_id);
}

void shield_id (int ashield_id)
{       ROSE_PUT_PRIM (int,PERSISTENT_shield_id,ashield_id); }

/* pin_pair_index Access Methods */
int pin_pair_index()
{       return ROSE_GET_PRIM (int,PERSISTENT_pin_pair_index);
}

void pin_pair_index (int apin_pair_index)
{       ROSE_PUT_PRIM (int,PERSISTENT_pin_pair_index,apin_pair_index); }

/* pin_pair Access Methods */
pin_pair_rec * pin_pair()
{       return ROSE_GET_OBJ (pin_pair_rec,PERSISTENT_pin_pair);
}

void pin_pair (pin_pair_rec * apin_pair)
{       ROSE_PUT_OBJ (pin_pair_rec,PERSISTENT_pin_pair,apin_pair); }

/* ww_data Access Methods */
ww_route_data_rec * ww_data()
{       return ROSE_GET_OBJ (ww_route_data_rec,PERSISTENT_ww_data);
}

void ww_data (ww_route_data_rec * aww_data)
{       ROSE_PUT_OBJ (ww_route_data_rec,PERSISTENT_ww_data,aww_data); }

/* comment Access Methods */
STR comment()
{       return ROSE_GET_PRIM (STR,PERSISTENT_comment);
}

void comment (STR acomment)
{       ROSE_PUT_PRIM (STR,PERSISTENT_comment,acomment); }
```

```
/* Constructors */
route_rec ();
route_rec (
        STR asignal,
        STR aroute_type,
        STR astatus,
        pin_name_rec * atarget_name,
        pin_name_rec * aobject_name,
        pin_rec * atarget_pin,
        pin_rec * aobject_pin,
        point_rec * atarget_loc,
        point_rec * aobject_loc,
        BOOL aprotect,
        int atarget_layer,
        int aobject_layer,
        ListOfsegment_rec * apath,
        int ashield_id,
        int apin_pair_index,
        pin_pair_rec * apin_pair,
        ww_route_data_rec * aww_data,
        STR acomment );
};
```

5.2 ProcessModel

The ProcessModel class is used to manage the manufacturing process models. Each ProcessModel object contains a reference to the top node in the process dependency graph. Each also contains the name of the model, the dates of its creation and last modification, and the name of the author of the model. The ProcessModel objects will be created by the Modeler managing object. The Analyzer will traverse the ProcessModel in order to select the appropriate analysis plan for the ProductDesign under analysis, and calculate the corresponding yield, rework, and cost of each selected process and operation. The analysis plan is a subset of the original ProcessModel object. The Advisor managing object will provide viewing of the resulting Analyzer process plan. The Class Diagrams for the Process Model, Resources, and ComplexRules are illustrates in Figures 5.1-1, 5.1-2, and 5-1.3. The sub-sections that follow detail each of the ProcessModel, Resource, and ComplexRule classes/objects and their corresponding methods.

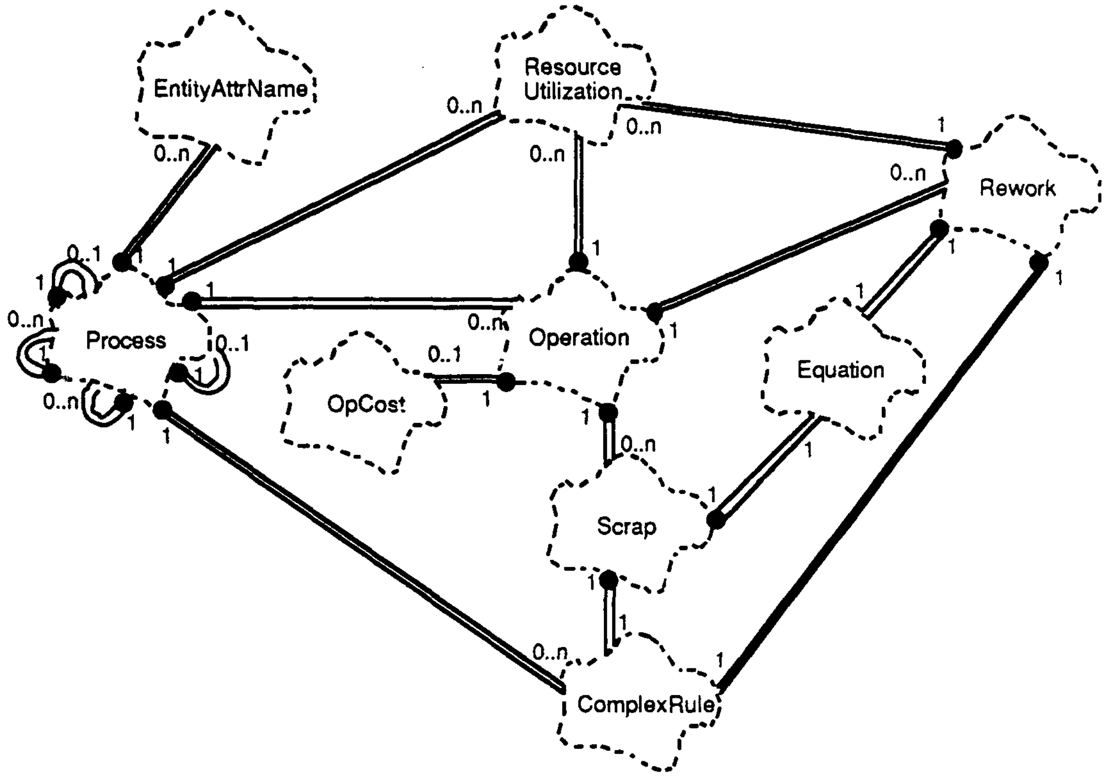


Figure 5.2-1 Process Model Class Diagram

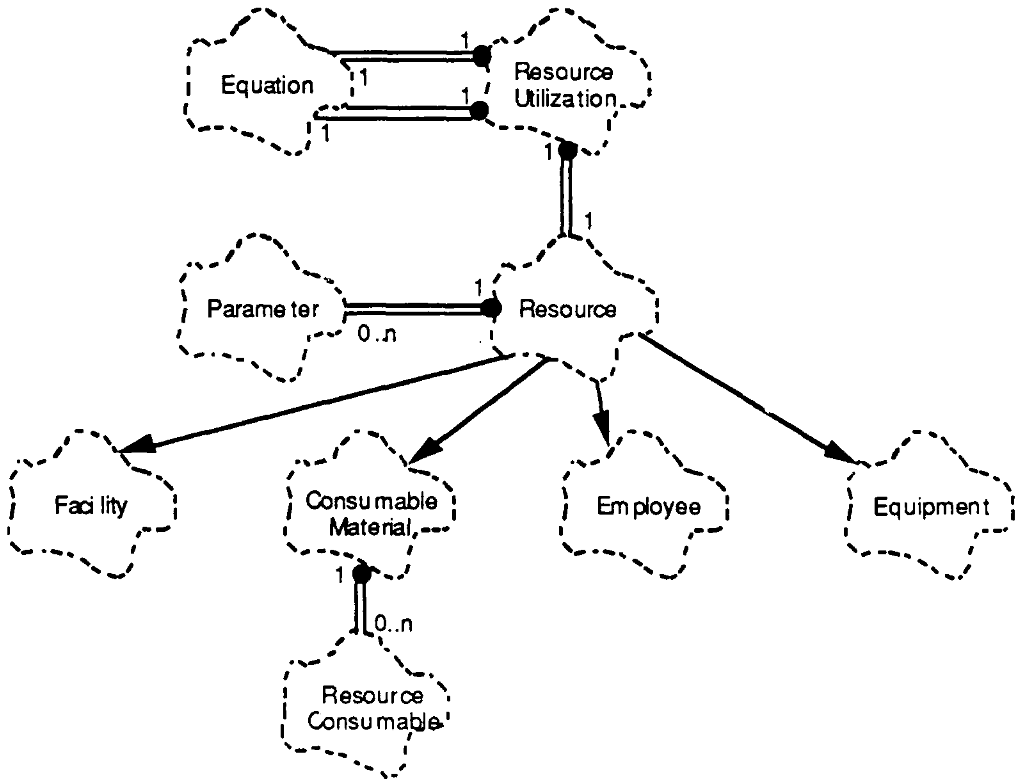


Figure 5.2-2 Resource Class Diagram

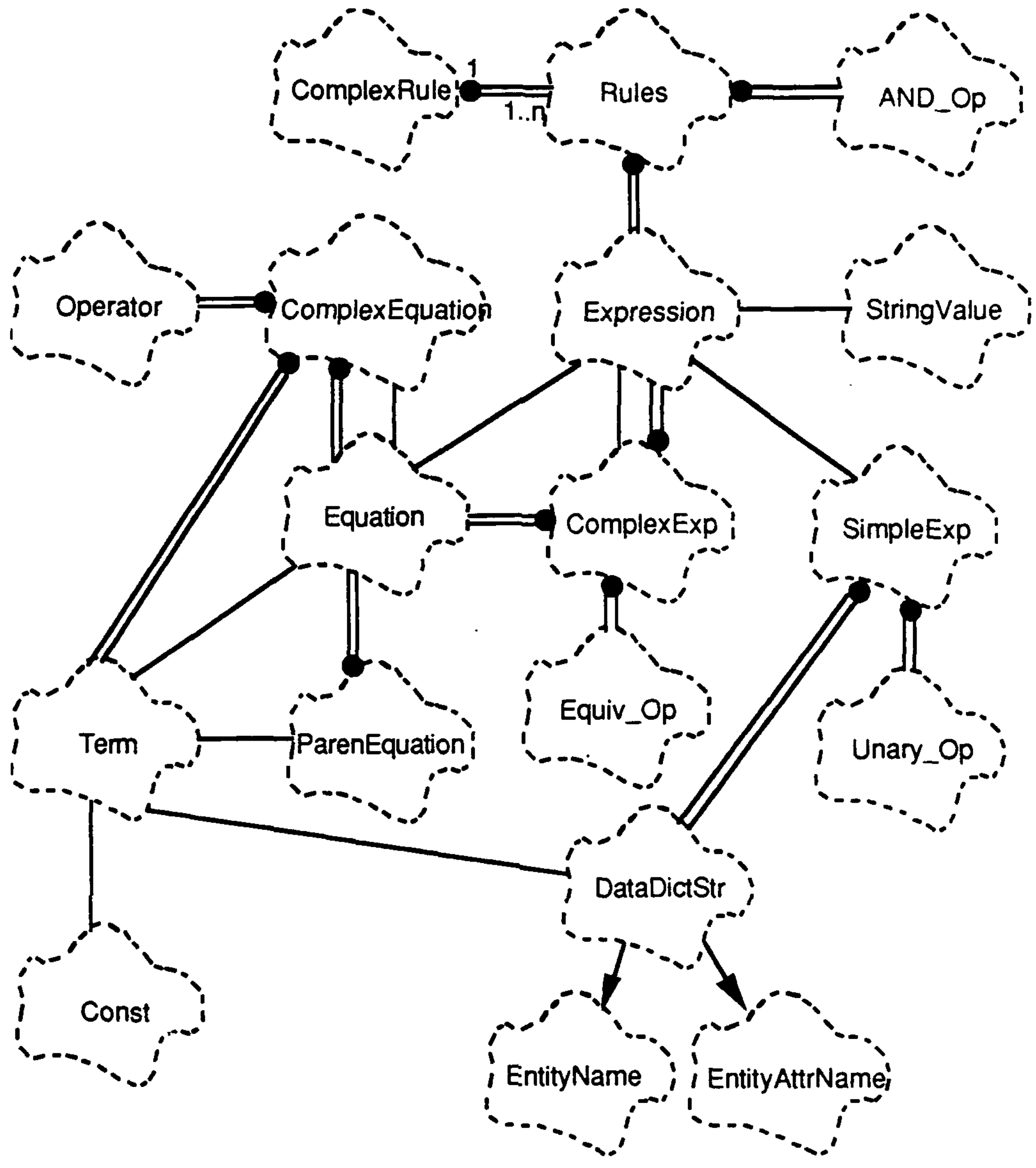


Figure 5.2-3 ComplexRule Class Diagram

5.2.1 ProcessModel Specification

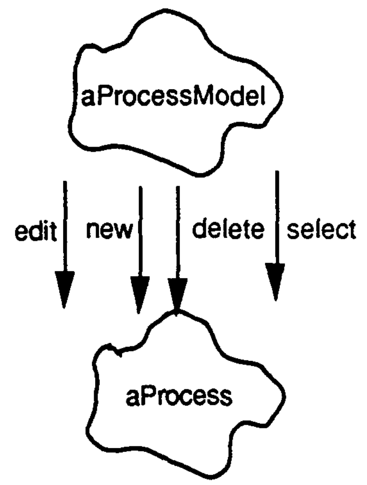


Figure 5.2-4 ProcessModel Object Diagram

```
/* Class Declaration */
ROSE_DECLARE (ProcessModel) : virtual public RoseStructure {
 private:
            STR PERSISTENT_name;
            Date * PERSISTENT_creationDate;
            Date * PERSISTENT_modifyDate;
            STR PERSISTENT_author;
            Process * PERSISTENT_topProcess;

 public:
            ROSE_DECLARE_MEMBERS(ProcessModel);

/* Access and Update Methods */
/* name Access Methods */
STR name()
{           return ROSE_GET_PRIM (STR,PERSISTENT_name);
}
void name (STR aname)
{           ROSE_PUT_PRIM (STR,PERSISTENT_name,aname); }

/* creationDate Access Methods */
Date * creationDate()
{           return ROSE_GET_OBJ (Date,PERSISTENT_creationDate);
}
void creationDate (Date * acreationDate)
{           ROSE_PUT_OBJ (Date,PERSISTENT_creationDate,acreationDate); }

/* modifyDate Access Methods */
Date * modifyDate()
{           return ROSE_GET_OBJ (Date,PERSISTENT_modifyDate);
}
void modifyDate (Date * amodifyDate)
{           ROSE_PUT_OBJ (Date,PERSISTENT_modifyDate,amodifyDate); }

/* author Access Methods */
STR author()
{           return ROSE_GET_PRIM (STR,PERSISTENT_author);
}
void author (STR aauthor)
{           ROSE_PUT_PRIM (STR,PERSISTENT_author,aauthor); }

/* topProcess Access Methods */
Process * topProcess()
{           return ROSE_GET_OBJ (Process,PERSISTENT_topProcess);
}
void topProcess (Process * atopProcess)
{           ROSE_PUT_OBJ (Process,PERSISTENT_topProcess,atopProcess); }

/* Constructors */
ProcessModel ();
ProcessModel (
            STR aname,
            Date *  acreationDate,
            Date *  amodifyDate,
            STR aauthor,
            Process *  atopProcess );
};
ProcessModel::newModel ( );
ProcessModel::selectModel( );
ProcessModel::deleteModel ( );
ProcessModel::saveModel ( );
ProcessModel::selectModel();

/* Methods Implementation */
ProcessModel::ProcessModel () {
            PERSISTENT_name = NULL;
            PERSISTENT_creationDate = NULL; PERSISTENT_modifyDate =
            NULL;
            PERSISTENT_author = NULL;
            PERSISTENT_topProcess = NULL;
            ROSE_CTOR_EXTENSIONS;
}

ProcessModel::ProcessModel (
            STR aname,
            Date *  acreationDate,
            Date *  amodifyDate,
            STR aauthor,
            Process *  atopProcess )
{
            name (aname);
            creationDate (acreationDate);
            modifyDate (amodifyDate);
            author (aauthor);
            topProcess (atopProcess);
            ROSE_CTOR_EXTENSIONS;
}
```

```
/* Additional Methods */
ProcessModel::newModel ( ) {}
ProcessModel::selectModel( ) {}
ProcessModel::deleteModel ( ) {}
ProcessModel::saveModel ( ) {}
ProcessModel::selectModel ( ) {}
```

5.2.2 Process Specification

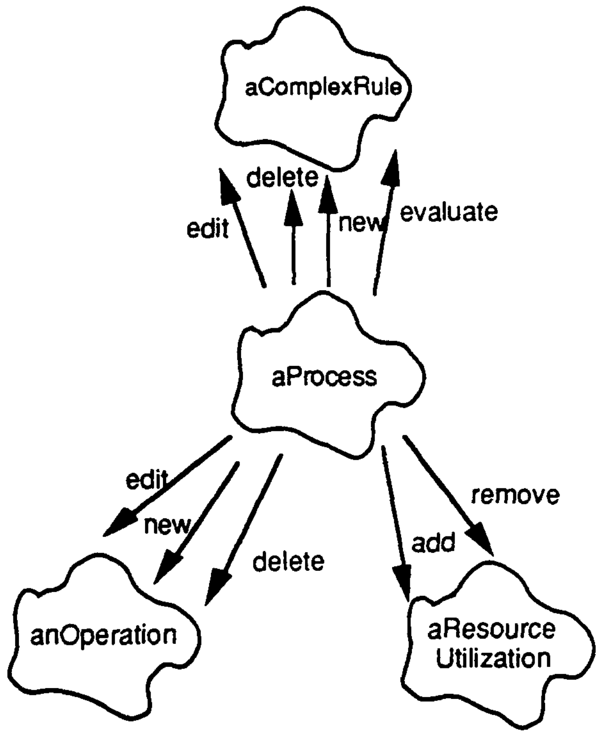


Figure 5.2-5 Process Object Diagram

```
/* Class Declaration */
ROSE_DECLARE (Process) : virtual public RoseStructure {
 private:
        STR PERSISTENT_name;
        BOOL PERSISTENT_andParents;
        ListOfComplexRule * PERSISTENT_rules;
        ListOfComplexRule * PERSISTENT_elimrules;
        ListOfResource * PERSISTENT_resources;
        ListOfRoseObject * PERSISTENT_parents;
        ListOfRoseObject * PERSISTENT_children;
        RoseObject * PERSISTENT_lsibling;
        RoseObject * PERSISTENT_rsibling;
        ListOfOperation * PERSISTENT_operList;
        ListOfEntityAttrName * PERSISTENT_designFeatures;

 public:
        ROSE_DECLARE_MEMBERS(Process);

/* Access and Update Methods */
/* name Access Methods */
STR name()
{       return ROSE_GET_PRIM (STR,PERSISTENT_name);
}
void name (STR aname)
{       ROSE_PUT_PRIM (STR,PERSISTENT_name,aname); }

/* andParents Access Methods */
BOOL andParents()
{       return ROSE_GET_PRIM (BOOL,PERSISTENT_andParents);
}
void andParents (BOOL aandParents)
{       ROSE_PUT_PRIM (BOOL,PERSISTENT_andParents,aandParents); }

/* rules Access Methods */
ListOfComplexRule * rules();
void rules (ListOfComplexRule *  arules)
{       ROSE_PUT_OBJ (ListOfComplexRule,PERSISTENT_rules,arules); }

/* elimrules Access Methods */
ListOfComplexRule * elimrules();
void elimrules (ListOfComplexRule *  aelimrules)
{       ROSE_PUT_OBJ (ListOfComplexRule,PERSISTENT_elimrules,aelimrules); }

/* resources Access Methods */
ListOfResource * resources();
void resources (ListOfResource *  aresources)
{       ROSE_PUT_OBJ (ListOfResource,PERSISTENT_resources,aresources); }

/* parents Access Methods */
ListOfRoseObject * parents();
void parents (ListOfRoseObject *  aparents)
{       ROSE_PUT_OBJ (ListOfRoseObject,PERSISTENT_parents,aparents); }

/* children Access Methods */
ListOfRoseObject * children();
void children (ListOfRoseObject *  achildren)
{       ROSE_PUT_OBJ (ListOfRoseObject,PERSISTENT_children,achildren); }

/* lsibling Access Methods */
RoseObject * lsibling()
  { return ROSE_GET_OBJ (RoseObject,PERSISTENT_lsibling); }
void lsibling (RoseObject * alsibling)
{       ROSE_PUT_OBJ(RoseObject,PERSISTENT_lsibling,alsibling); }

/* rsibling Access Methods */
RoseObject * rsibling()
  { return ROSE_GET_OBJ (RoseObject,PERSISTENT_rsibling); }
void rsibling (RoseObject * arsibling)
{       ROSE_PUT_OBJ(RoseObject,PERSISTENT_rsibling,arsibling); }

/* operList Access Methods */
ListOfOperation * operList();
void operList (ListOfOperation *  aoperList)
```

```
{       ROSE_PUT_OBJ (ListOfOperation,PERSISTENT_operList,aoperList); }

/* designFeatures Access Methods */
ListOfEntityAttrName * designFeatures();
void designFeatures (ListOfEntityAttrName *  adesignFeatures)
{       ROSE_PUT_OBJ
(ListOfEntityAttrName,PERSISTENT_designFeatures,adesignFeatures); }

/* Constructors */
Process ();
Process (
        STR aname,
        BOOL aandParents,
        ListOfComplexRule *  arules,
        ListOfComplexRule *  aelimrules,
        ListOfResource *  aresources,
        ListOfRoseObject *  aparents,
        ListOfRoseObject *  achildren,
        RoseObject *  alsibling,
        RoseObject *  arsibling,
        ListOfOperation *  aoperList,
        ListOfEntityAttrName *  adesignFeatures );

/* CLASS DECLARATION EXTENSIONS */
void new( );
void edit( );
void delete( );
int select();
```

```
/* Methods Implementation */
Process::Process () {
        PERSISTENT_name = NULL;
        PERSISTENT_andParents = FALSE;
        PERSISTENT_rules = NULL;
        PERSISTENT_elimrules = NULL;
        PERSISTENT_resources = NULL;
        PERSISTENT_parents = NULL;
        PERSISTENT_children = NULL;
        PERSISTENT_lsibling = NULL;
        PERSISTENT_rsibling = NULL;
        PERSISTENT_operList = NULL;
        PERSISTENT_designFeatures = NULL;
        ROSE_CTOR_EXTENSIONS;
}

Process::Process (
        STR aname,
        BOOL aandParents,
        ListOfComplexRule *  arules,
        ListOfComplexRule *  aelimrules,
        ListOfResource *  aresources,
        ListOfRoseObject *  aparents,
        ListOfRoseObject *  achildren,
        RoseObject *  alsibling,
        RoseObject * arsibling,
        ListOfOperation * aoperList,
        ListOfEntityAttrName * adesignFeatures )
{
        name (aname);
        andParents (aandParents);
        rules (arules);
        elimrules (aelimrules);
        resources (aresources);
        parents (aparents);
        children (achildren);
        lsibling (alsibling);
        rsibling (arsibling);
        operList (aoperList);
        designFeatures (adesignFeatures);
        ROSE_CTOR_EXTENSIONS;
}

ListOfComplexRule * Process :: rules()
{       if( !PERSISTENT_rules)
        if( this->isPersistent())
                rules (pnewIn (design()) ListOfComplexRule);
        else    rules (new ListOfComplexRule);
        return ROSE_GET_OBJ (ListOfComplexRule,PERSISTENT_rules);
}

ListOfComplexRule * Process :: elimrules()
{       if( !PERSISTENT_elimrules)
        if( this->isPersistent())
                elimrules (pnewIn (design()) ListOfComplexRule);
        else    elimrules (new ListOfComplexRule);
        return ROSE_GET_OBJ (ListOfComplexRule,PERSISTENT_elimrules);
}

ListOfResource * Process :: resources()
{       if( !PERSISTENT_resources)
        if( this->isPersistent())
                resources (pnewIn (design()) ListOfResource);
        else    resources (new ListOfResource);
        return ROSE_GET_OBJ (ListOfResource,PERSISTENT_resources);
}

ListOfRoseObject * Process :: parents()
{       if( !PERSISTENT_parents)
        if( this->isPersistent())
                parents (pnewIn (design()) ListOfRoseObject);
        else    parents (new ListOfRoseObject);
        return ROSE_GET_OBJ (ListOfRoseObject,PERSISTENT_parents);
}

ListOfRoseObject * Process :: children()
{       if( !PERSISTENT_children)
        if( this->isPersistent())
                children (pnewIn (design()) ListOfRoseObject);
        else    children (new ListOfRoseObject);
        return ROSE_GET_OBJ (ListOfRoseObject,PERSISTENT_children);
}

ListOfOperation * Process :: operList()
{       if( !PERSISTENT_operList)
        if( this->isPersistent())
                operList (pnewIn (design()) ListOfOperation);
        else    operList (new ListOfOperation);
        return ROSE_GET_OBJ (ListOfOperation,PERSISTENT_operList);
}

ListOfEntityAttrName * Process :: designFeatures()
{       if( !PERSISTENT_designFeatures)
        if( this->isPersistent())
                designFeatures (pnewIn (design()) ListOfEntityAttrName);
        else    designFeatures (new ListOfEntityAttrName);
        return ROSE_GET_OBJ (ListOfEntityAttrName,PERSISTENT_designFeatures);
}

/* Additional  Methods */
void Process::new( ) {}
void Process::edit( ) {}
void Process::delete( ) {}
int Process::select() {}
```

5.2.3 Operation Specification

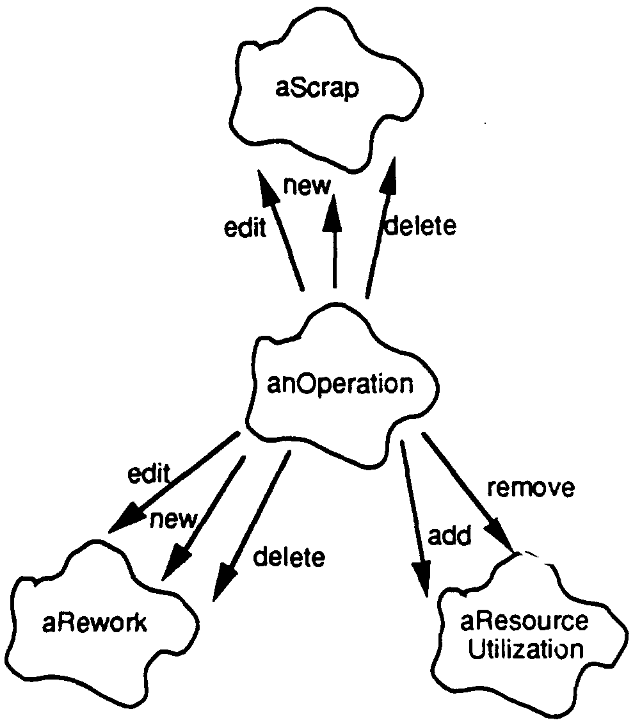


Figure 5.2-6 Operation Object Diagram

```
/* Class Declaration */
ROSE_DECLARE (Operation) : virtual public RoseStructure {
 private:
        STR PERSISTENT_name;
        STR PERSISTENT_desc;
        ListOfResourceUtilization * PERSISTENT_resources;
        ListOfScrap * PERSISTENT_scrap_rate;
        ListOfRework * PERSISTENT_rework_rate;
        OpCost * PERSISTENT_cost;       /* OPTIONAL */

 public:
        ROSE_DECLARE_MEMBERS(Operation);

/* Access and Update Methods */
/* name Access Methods */
STR name()
{       return ROSE_GET_PRIM (STR,PERSISTENT_name);
}
void name (STR aname)
{       ROSE_PUT_PRIM (STR,PERSISTENT_name,aname); }

/* desc Access Methods */
STR desc()
{       return ROSE_GET_PRIM (STR,PERSISTENT_desc);
}
void desc (STR adesc)
{       ROSE_PUT_PRIM (STR,PERSISTENT_desc,adesc); }

/* resources Access Methods */
ListOfResourceUtilization * resources();
void resources (ListOfResourceUtilization * aresources)
{       ROSE_PUT_OBJ (ListOfResourceUtilization,PERSISTENT_resources,aresources); }

/* scrap_rate Access Methods */
ListOfScrap * scrap_rate();
void scrap_rate (ListOfScrap * ascrap_rate)
{       ROSE_PUT_OBJ (ListOfScrap,PERSISTENT_scrap_rate,ascrap_rate); }

/* rework_rate Access Methods */
ListOfRework * rework_rate();
void rework_rate (ListOfRework * arework_rate)
{       ROSE_PUT_OBJ (ListOfRework,PERSISTENT_rework_rate,arework_rate); }

/* cost Access Methods */
OpCost * cost()
{       return ROSE_GET_OBJ (OpCost,PERSISTENT_cost);
}
void cost (OpCost * acost)
{       ROSE_PUT_OBJ (OpCost,PERSISTENT_cost,acost); }

/* Constructors */
Operation ();
Operation (
        STR aname,
        STR adesc,
        ListOfResourceUtilization *  aresources,
        ListOfScrap *  ascrap_rate,
        ListOfRework *  arework_rate,
        OpCost *  acost );

/* CLASS DECLARATION EXTENSIONS */
void new( );
void edit( );
void delete( );

};

/* Methods Implementation */
Operation::Operation () {
        PERSISTENT_name = NULL;
        PERSISTENT_desc = NULL;
        PERSISTENT_resources = NULL;
        PERSISTENT_scrap_rate = NULL;
        PERSISTENT_rework_rate = NULL;
        PERSISTENT_cost = NULL;
        ROSE_CTOR_EXTENSIONS;
}

Operation::Operation (
        STR aname,
        STR adesc,
        ListOfResourceUtilization *  aresources,
        ListOfScrap *  ascrap_rate,
        ListOfRework *  arework_rate,
        OpCost *  acost )
{
        name (aname);
        desc (adesc);
        resources (aresources);
        scrap_rate (ascrap_rate);
        rework_rate (arework_rate);
        cost (acost);
        ROSE_CTOR_EXTENSIONS;
}

ListOfResourceUtilization * Operation :: resources()
{       if( !PERSISTENT_resources)
        if( this->isPersistent())
                resources (pnewIn (design()) ListOfResourceUtilization);
        else    resources (new ListOfResourceUtilization);
        return ROSE_GET_OBJ (ListOfResourceUtilization,PERSISTENT_resources);
}

ListOfScrap * Operation :: scrap_rate()
{       if( !PERSISTENT_scrap_rate)
        if( this->isPersistent())
                scrap_rate (pnewIn (design()) ListOfScrap);
        else    scrap_rate (new ListOfScrap);
        return ROSE_GET_OBJ (ListOfScrap,PERSISTENT_scrap_rate);
}
```

ListOfRework * Operation :: rework_rate()

```
{       if( !PERSISTENT_rework_rate)
        if( this->isPersistent())
                rework_rate (pnewIn (design()) ListOfRework);
        else    rework_rate (new ListOfRework);
        return ROSE_GET_OBJ (ListOfRework,PERSISTENT_rework_rate);
}
```

void Operation::new( ) {}
void Operation::edit( ) {}
void Operation::delete( ) {}

5.2.4 Scrap Specification

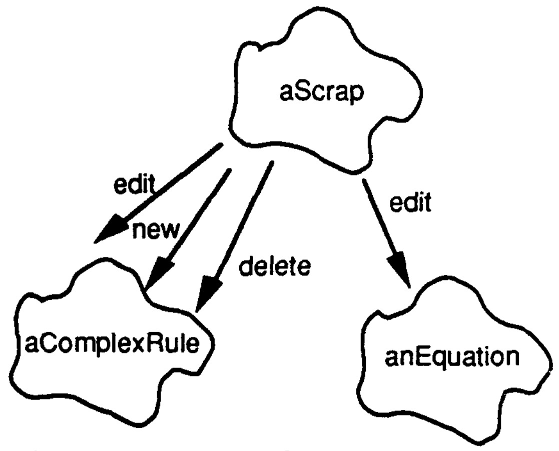


Figure 5.2-7 Scrap Object Diagram

/* Class Declaration */
ROSE_DECLARE (**Scrap**) : virtual public RoseStructure {

```
 private:
        ComplexExp *  PERSISTENT_scrapRule;
        Equation *  PERSISTENT_scrapRate;

 public:
        ROSE_DECLARE_MEMBERS(Scrap);
```

/* Access and Update Methods */

```
/* scrapRule Access Methods */
ComplexExp * scrapRule()
{       return ROSE_GET_OBJ (ComplexExp,PERSISTENT_scrapRule);
}
void scrapRule (ComplexExp * ascrapRule)
{       ROSE_PUT_OBJ (ComplexExp,PERSISTENT_scrapRule,ascrapRule); }

/* scrapRate Access Methods */
Equation * scrapRate()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_scrapRate); }
```

```
void scrapRate (Equation * ascrapRate)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_scrapRate,ascrapRate); }

/* Constructors */
Scrap ();
Scrap (
        ComplexExp *  ascrapRule,
        Equation *  ascrapRate );

/* CLASS DECLARATION EXTENSIONS */

void new( );
void edit( );
void delete( );
};
```

```
/* Methods Implementation */
Scrap::Scrap () {
        PERSISTENT_scrapRule = NULL;
        PERSISTENT_scrapRate = NULL;
        ROSE_CTOR_EXTENSIONS;
}

Scrap::Scrap (
        ComplexExp *  ascrapRule,
        Equation *  ascrapRate )
{
        scrapRule (ascrapRule);
        scrapRate (ascrapRate);
        ROSE_CTOR_EXTENSIONS;
}

void Scrap::new( ) {}
void Scrap::edit( ) {}
void Scrap::delete( ) {}
```

5.2.5 Rework Specification

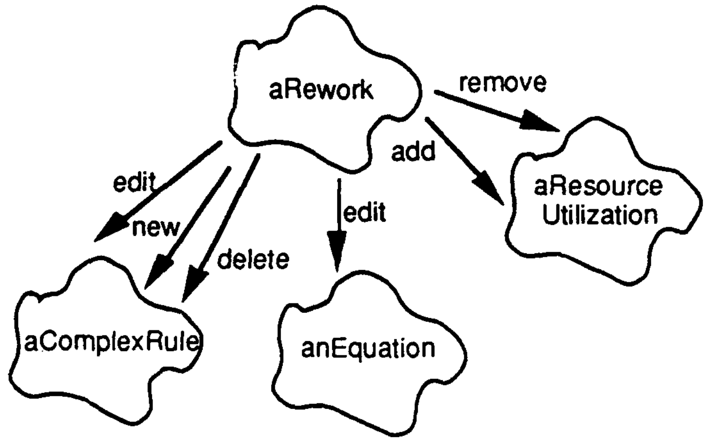


Figure 5.2-8 Rework Object Diagram

```
/* Class De  aration */
ROSE_DEC_ARE (Rework) : virtual public RoseStructure {
 private:
        ComplexExp *  PERSISTENT_reworkRule;
        Equation *  PERSISTENT_reworkRate;
        ListOfResource *  PERSISTENT_resources;

 public:
        ROSE_DECLARE_MEMBERS(Rework);

/* Access and Update Methods */
/* reworkRule Access Methods */
ComplexExp * reworkRule()
{       return ROSE_GET_OBJ (ComplexExp,PERSISTENT_reworkRule);
}
void reworkRule (ComplexExp * areworkRule)
{       ROSE_PUT_OBJ (ComplexExp,PERSISTENT_reworkRule,areworkRule); }

/* reworkRate Access Methods */
Equation * reworkRate()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_reworkRate); }
void reworkRate (Equation * areworkRate)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_reworkRate,areworkRate); }

/* resources Access Methods */
ListOfResource * resources();
void resources (ListOfResource *  aresources)
{       ROSE_PUT_OBJ (ListOfResource,PERSISTENT_resources,aresources); }

/* Constructors */
Rework ();
Rework (
        ComplexExp *  areworkRule,
        Equation *  areworkRate,
        ListOfResource *  aresources );

void new( );
void edit( );
void delete( );
};

/* Methods Implementation */
Rework::Rework () {
        PERSISTENT_reworkRule = NULL;
        PERSISTENT_reworkRate = NULL;
        PERSISTENT_resources = NULL;
        ROSE_CTOR_EXTENSIONS;
}

Rework::Rework (
        ComplexExp *  areworkRule,
        Equation *  areworkRate,
        ListOfResource *  aresources )
{
        reworkRule (areworkRule);
        reworkRate (areworkRate);
        resources (aresources);
        ROSE_CTOR_EXTENSIONS;
}
```

ListOfResource * Rework :: resources()

```
{       if( !PERSISTENT_resources)
        if( this->isPersistent())
                resources (pnewIn (design()) ListOfResource);
        else    resources (new ListOfResource);
        return ROSE_GET_OBJ (ListOfResource,PERSISTENT_resources);
}
```

void Rework::new( ) {}
void Rework::edit( ) {}
void Rework::delete( ) {}

5.2.6 OpCost Specification

/* **Class Declaration** */

```
ROSE_DECLARE (OpCost) : virtual public RoseStructure {
 private:
        float PERSISTENT_setupTime;
        float PERSISTENT_runTime;
        float PERSISTENT_scrapPercentage;
        float PERSISTENT_reworkPercentage;
        float PERSISTENT_reworkCost;
        int PERSISTENT_prodQty;
        float PERSISTENT_idealFAIT;
        float PERSISTENT_actualFAIT;

 public:
        ROSE_DECLARE_MEMBERS(OpCost);
```

/* **Access and Update Methods** */

```
/* setupTime Access Methods */
float setupTime()
{       return ROSE_GET_PRIM (float,PERSISTENT_setupTime);
}
void setupTime (float asetupTime)
{       ROSE_PUT_PRIM (float,PERSISTENT_setupTime,asetupTime); }

/* runTime Access Methods */
float runTime()
{       return ROSE_GET_PRIM (float,PERSISTENT_runTime);
}
void runTime (float arunTime)
{       ROSE_PUT_PRIM (float,PERSISTENT_runTime,arunTime); }

/* scrapPercentage Access Methods */
float scrapPercentage()
{       return ROSE_GET_PRIM (float,PERSISTENT_scrapPercentage);
}
void scrapPercentage (float ascrapPercentage)
{       ROSE_PUT_PRIM (float,PERSISTENT_scrapPercentage,ascrapPercentage); }

/* reworkPercentage Access Methods */
float reworkPercentage()
{       return ROSE_GET_PRIM (float,PERSISTENT_reworkPercentage);
}
void reworkPercentage (float areworkPercentage)
{       ROSE_PUT_PRIM (float,PERSISTENT_reworkPercentage,areworkPercentage); }

/* reworkCost Access Methods */
float reworkCost()
{       return ROSE_GET_PRIM (float,PERSISTENT_reworkCost);
}
void reworkCost (float areworkCost)
{       ROSE_PUT_PRIM (float,PERSISTENT_reworkCost,areworkCost); }

/* prodQty Access Methods */
int prodQty()
{       return ROSE_GET_PRIM (int,PERSISTENT_prodQty);
}
void prodQty (int aprodQty)
{       ROSE_PUT_PRIM (int,PERSISTENT_prodQty,aprodQty); }

/* idealFAIT Access Methods */
float idealFAIT()
{       return ROSE_GET_PRIM (float,PERSISTENT_idealFAIT);
}
void idealFAIT (float aidealFAIT)
{       ROSE_PUT_PRIM (float,PERSISTENT_idealFAIT,aidealFAIT); }

/* actualFAIT Access Methods */
float actualFAIT()
{       return ROSE_GET_PRIM (float,PERSISTENT_actualFAIT);
}
void actualFAIT (float aactualFAIT)
{       ROSE_PUT_PRIM (float,PERSISTENT_actualFAIT,aactualFAIT); }

/* Constructors */
OpCost ();
OpCost (
        float asetupTime,
        float arunTime,
        float ascrapPercentage,
        float areworkPercentage,
        float areworkCost,
        int aprodQty,
        float aidealFAIT,
        float aactualFAIT );

};
```

/* Methods Implementation */

```
OpCost::OpCost () {
        PERSISTENT_setupTime = 0;
        PERSISTENT_runTime = 0;
        PERSISTENT_scrapPercentage = 0;
        PERSISTENT_reworkPercentage = 0;
        PERSISTENT_reworkCost = 0;
        PERSISTENT_prodQty = 0;
        PERSISTENT_idealFAIT = 0;
        PERSISTENT_actualFAIT = 0;
        ROSE_CTOR_EXTENSIONS;
}

OpCost::OpCost (
        float asetupTime,
        float arunTime,
        float ascrapPercentage,
        float areworkPercentage,
        float areworkCost,
        int aprodQty,
        float aidealFAIT,
        float aactualFAIT )
{
        setupTime (asetupTime);
        runTime (arunTime);
        scrapPercentage (ascrapPercentage);
        reworkPercentage (areworkPercentage);
        reworkCost (areworkCost);
        prodQty (aprodQty);
        idealFAIT (aidealFAIT);
        actualFAIT (aactualFAIT);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.7 ResourceUtilization Specification

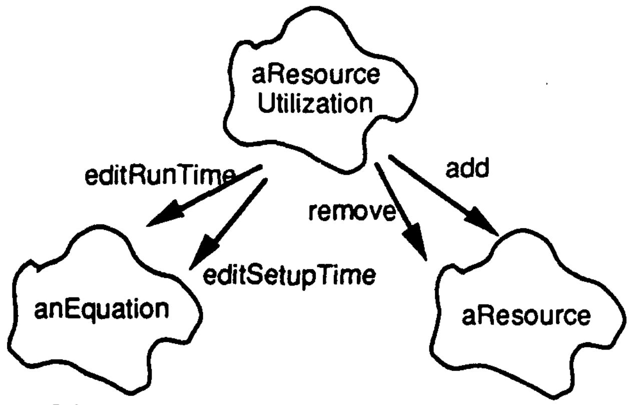


Figure 5.2-9 ResourceUtilization Object Diagram

```
/* Class Declaration */
ROSE_DECLARE (ResourceUtilization) : virtual public RoseStructure {
 private:
        Resource *  PERSISTENT_resource;
        RoseObject *  PERSISTENT_setupTime;
        RoseObject *  PERSISTENT_runTime;
        float PERSISTENT_effRate;  /* OPTIONAL */

  public:
        ROSE_DECLARE_MEMBERS(ResourceUtilization);

/* Access and Update Methods */
/* resource Access Methods */
Resource * resource()
{       return ROSE_GET_OBJ (Resource,PERSISTENT_resource);
}
void resource (Resource * aresource)
{       ROSE_PUT_OBJ (Resource,PERSISTENT_resource,aresource); }

/* setupTime Access Methods */
RoseObject * setupTime()
  { return ROSE_GET_OBJ (RoseObject,PERSISTENT_setupTime); }
void setupTime (RoseObject * asetupTime)
{       ROSE_PUT_OBJ(RoseObject,PERSISTENT_setupTime,asetupTime); }

/* runTime Access Methods */
RoseObject * runTime()
  { return ROSE_GET_OBJ (RoseObject,PERSISTENT_runTime); }
void runTime (RoseObject * arunTime)
{       ROSE_PUT_OBJ(RoseObject,PERSISTENT_runTime,arunTime); }

/* effRate Access Methods */
float effRate()
{       return ROSE_GET_PRIM (float,PERSISTENT_effRate);
}
void effRate (float aeffRate)
{       ROSE_PUT_PRIM (float,PERSISTENT_effRate,aeffRate); }

/* Constructors */
ResourceUtilization ();
ResourceUtilization (
        Resource *  aresource,
        RoseObject *  asetupTime,
        RoseObject *  arunTime,
        float aeffRate );

/* CLASS DECLARATION EXTENSIONS */
void add( );
void remove( );
};

/* Methods Implementation */
ResourceUtilization::ResourceUtilization () {
        PERSISTENT_resource = NULL;
        PERSISTENT_setupTime = NULL;
        PERSISTENT_runTime = NULL;
        PERSISTENT_effRate = 0;
        ROSE_CTOR_EXTENSIONS;
```

```
}

ResourceUtilization::ResourceUtilization (
        Resource * aresource,
        RoseObject * asetupTime,
        RoseObject * arunTime,
        float aeffRate )
{
        resource (aresource);
        setupTime (asetupTime);
        runTime (arunTime);
        effRate (aeffRate);
        ROSE_CTOR_EXTENSIONS;
}

void ResourceUtilization::add( ) {}
void ResourceUtilization::remove( ) {}
```

5.2.8 Parameter Specification

```
/* Class Declaration */
ROSE_DECLARE (Parameter) : virtual public RoseStructure {
 private:
        STR PERSISTENT_p_name;
        STR PERSISTENT_p_value;

 public:
        ROSE_DECLARE_MEMBERS(Parameter);

/* Access and Update Methods */
/* p_name Access Methods */
STR p_name()
{       return ROSE_GET_PRIM (STR,PERSISTENT_p_name);
}
void p_name (STR ap_name)
{       ROSE_PUT_PRIM (STR,PERSISTENT_p_name,ap_name); }

/* p_value Access Methods */
STR p_value()
{       return ROSE_GET_PRIM (STR,PERSISTENT_p_value);
}
void p_value (STR ap_value)
{       ROSE_PUT_PRIM (STR,PERSISTENT_p_value,ap_value); }

/* Constructors */
Parameter ();
Parameter (
        STR ap_name,
        STR ap_value );
};

/* Methods Implementation */
Parameter::Parameter () {
        PERSISTENT_p_name = NULL;
        PERSISTENT_p_value = NULL;
        ROSE_CTOR_EXTENSIONS;
}
```

Parameter::Parameter (
STR ap_name,
STR ap_value)

```
{
        p_name (ap_name);
        p_value (ap_value);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.9 Resource Specification

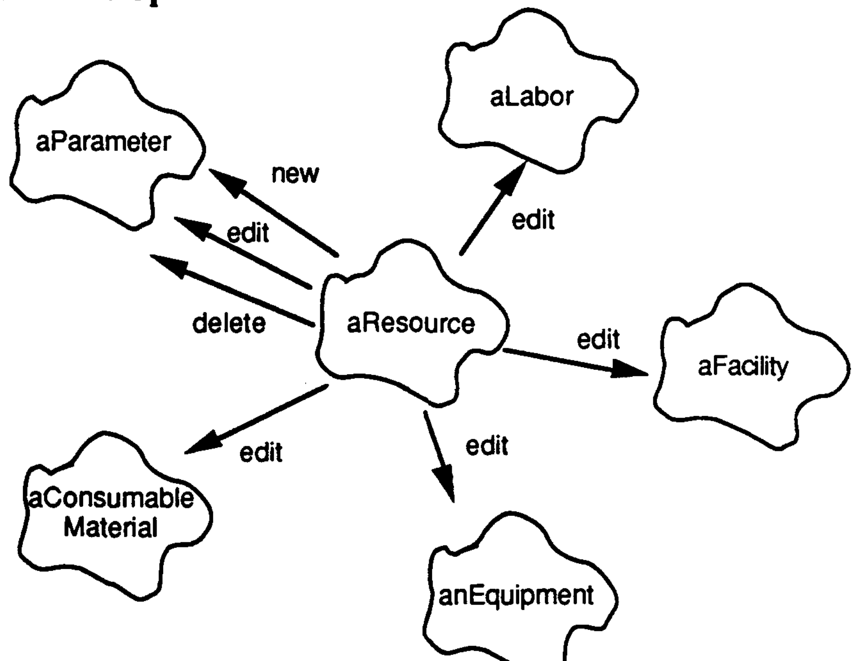


Figure 5.2-10 Resource Object Diagram

/* Class Declaration */
ROSE_DECLARE (**Resource**) : virtual public RoseStructure {

```
 private:
        STR PERSISTENT_resource_name;
        STR PERSISTENT_resource_code;
        ListOfParameter *  PERSISTENT_parameters;

 public:
        ROSE_DECLARE_MEMBERS(Resource);
```

/* Access and Update Methods */

```
/* resource_name Access Methods */
STR resource_name()
{       return ROSE_GET_PRIM (STR,PERSISTENT_resource_name);
}
```

```
void resource_name (STR aresource_name)
{       ROSE_PUT_PRIM (STR,PERSISTENT_resource_name,aresource_name); }

/* resource_code Access Methods */
STR resource_code()
{       return ROSE_GET_PRIM (STR,PERSISTENT_resource_code);
}
void resource_code (STR aresource_code)
{       ROSE_PUT_PRIM (STR,PERSISTENT_resource_code,aresource_code); }

/* parameters Access Methods */
ListOfParameter * parameters();
void parameters (ListOfParameter *  aparameters)
{       ROSE_PUT_OBJ (ListOfParameter,PERSISTENT_parameters,aparameters); }

/* Constructors */
Resource ();
Resource (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters );

/* CLASS DECLARATION EXTENSIONS */
void new( )
void edit( )
void delete( )
};

/* Methods Implementation */
Resource::Resource () {
        PERSISTENT_resource_name = NULL;
        PERSISTENT_resource_code = NULL;
        PERSISTENT_parameters = NULL;
        ROSE_CTOR_EXTENSIONS;
}

Resource::Resource (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters )
{
        resource_name (aresource_name);
        resource_code (aresource_code);
        parameters (aparameters);
        ROSE_CTOR_EXTENSIONS;
}

ListOfParameter * Resource :: parameters()
{       if( !PERSISTENT_parameters)
        if( this->isPersistent())
                parameters (pnewIn (design()) ListOfParameter);
        else    parameters (new ListOfParameter);
        return ROSE_GET_OBJ (ListOfParameter,PERSISTENT_parameters);
}
```

```
void Resource::new( ) {}
void Resource::edit( ) {}
void Resource::delete( ) {}
```

5.2.9.1 Equipment Specification

```
/* Class Declaration */
ROSE_DECLARE (Equipment) : virtual public Resource {
 private:
        STR PERSISTENT_equipment_category;
        float PERSISTENT_cost_per_time_unit;

 public:
        ROSE_DECLARE_MEMBERS(Equipment);

/* Access and Update Methods */
/* equipment_category Access Methods */
STR equipment_category()
{       return ROSE_GET_PRIM (STR,PERSISTENT_equipment_category);
}
void equipment_category (STR aequipment_category)
{       ROSE_PUT_PRIM (STR,PERSISTENT_equipment_category,aequipment_category);
}

/* cost_per_time_unit Access Methods */
float cost_per_time_unit()
{       return ROSE_GET_PRIM (float,PERSISTENT_cost_per_time_unit);
}
void cost_per_time_unit (float acost_per_time_unit)
{       ROSE_PUT_PRIM (float,PERSISTENT_cost_per_time_unit,acost_per_time_unit); }

/* Constructors */
Equipment ();
Equipment (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters,
        STR aequipment_category,
        float acost_per_time_unit );
};

/* Methods Implementation */
Equipment::Equipment () {
        PERSISTENT_equipment_category = NULL;
        PERSISTENT_cost_per_time_unit = 0;
        ROSE_CTOR_EXTENSIONS;
}

Equipment::Equipment (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters,
        STR aequipment_category,
        float acost_per_time_unit )
{
        resource_name (aresource_name);
        resource_code (aresource_code);
        parameters (aparameters);
        equipment_category (aequipment_category);
        cost_per_time_unit (acost_per_time_unit);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.9.2 ConsumableMaterial Specification

```
/* Class Declaration */
ROSE_DECLARE (ConsumableMaterial) : virtual public Resource {
 private:
        float PERSISTENT_cost_per_unit;
        ListOfResourceConsumable *  PERSISTENT_resourceRates;

 public:
        ROSE_DECLARE_MEMBERS(ConsumableMaterial);

/* Access and Update Methods */
/* cost_per_unit Access Methods */
float cost_per_unit()
{       return ROSE_GET_PRIM (float,PERSISTENT_cost_per_unit);
}
void cost_per_unit (float acost_per_unit)
{       ROSE_PUT_PRIM (float,PERSISTENT_cost_per_unit,acost_per_unit); }

/* resourceRates Access Methods */
ListOfResourceConsumable * resourceRates();
void resourceRates (ListOfResourceConsumable *  aresourceRates)
{       ROSE_PUT_OBJ
(ListOfResourceConsumable,PERSISTENT_resourceRates,aresourceRates); }

/* Constructors */
ConsumableMaterial ();
ConsumableMaterial (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters,
        float acost_per_unit,
        ListOfResourceConsumable *  aresourceRates );
};

/* Methods Implementation */
ConsumableMaterial::ConsumableMaterial () {
        PERSISTENT_cost_per_unit = 0;
        PERSISTENT_resourceRates = NULL;
        ROSE_CTOR_EXTENSIONS;
}
```

```
ConsumableMaterial::ConsumableMaterial (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter * aparameters,
        float acost_per_unit,
        ListOfResourceConsumable *  aresourceRates )
{
        resource_name (aresource_name);
        resource_code (aresource_code);
        parameters (aparameters);
        cost_per_unit (acost_per_unit);
        resourceRates (aresourceRates);
        ROSE_CTOR_EXTENSIONS;
}

ListOfResourceConsumable * ConsumableMaterial :: resourceRates()
{       if( !PERSISTENT_resourceRates)
        if( this->isPersistent())
                resourceRates (pnewIn (design()) ListOfResourceConsumable);
        else    resourceRates (new ListOfResourceConsumable);
        return ROSE_GET_OBJ (ListOfResourceConsumable,PERSISTENT_resourceRates);
}
```

5.2.9.3 ResourceConsumable Specification

```
/* Class Declaration */
ROSE_DECLARE (ResourceConsumable) : virtual public RoseStructure {
 private:
        Resource *  PERSISTENT_aresource;
        float PERSISTENT_units_exhausted_per_time_unit;

 public:
        ROSE_DECLARE_MEMBERS(ResourceConsumable);

/* Access and Update Methods */
/* aresource Access Methods */
Resource * aresource()
{       return ROSE_GET_OBJ (Resource,PERSISTENT_aresource);
}
void aresource (Resource * aaresource)
{       ROSE_PUT_OBJ (Resource,PERSISTENT_aresource,aaresource); }

/* units_exhausted_per_time_unit Access Methods */
float units_exhausted_per_time_unit()
{       return ROSE_GET_PRIM (float,PERSISTENT_units_exhausted_per_time_unit);
}
void units_exhausted_per_time_unit (float aunits_exhausted_per_time_unit)
{       ROSE_PUT_PRIM
(float,PERSISTENT_units_exhausted_per_time_unit,aunits_exhausted_per_time_unit); }

/* Constructors */
ResourceConsumable ();
ResourceConsumable (
        Resource *  aaresource,
        float aunits_exhausted_per_time_unit );
};
```

/* Methods Implementation */
ResourceConsumable::ResourceConsumable () {

```
        PERSISTENT_aresource = NULL;
        PERSISTENT_units_exhausted_per_time_unit = 0;
        ROSE_CTOR_EXTENSIONS;
}
```

ResourceConsumable::ResourceConsumable (
Resource * aaresource,
float aunits_exhausted_per_time_unit)

```
{
        aresource (aaresource);
        units_exhausted_per_time_unit (aunits_exhausted_per_time_unit);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.9.4 Labor Specification

/* Class Declaration */

```
ROSE_DECLARE (Labor) : virtual public Resource {
 private:
        STR PERSISTENT_job_code;
        float PERSISTENT_rate;

 public:
        ROSE_DECLARE_MEMBERS(Labor);
```

/* Access and Update Methods */

```
/* job_code Access Methods */
STR job_code()
{       return ROSE_GET_PRIM (STR,PERSISTENT_job_code);
}
void job_code (STR ajob_code)
{       ROSE_PUT_PRIM (STR,PERSISTENT_job_code,ajob_code); }

/* rate Access Methods */
float rate()
{       return ROSE_GET_PRIM (float,PERSISTENT_rate);
}
void rate (float arate)
{       ROSE_PUT_PRIM (float,PERSISTENT_rate,arate); }

/* Constructors */
Labor ();
Labor (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters,
        STR ajob_code,
        float arate );
};

/* Methods Implementation */
Labor::Labor () {
        PERSISTENT_job_code = NULL;
        PERSISTENT_rate = 0;
        ROSE_CTOR_EXTENSIONS;
}

Labor::Labor (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter * aparameters,
        STR ajob_code,
        float arate )
{
        resource_name (aresource_name);
        resource_code (aresource_code);
        parameters (aparameters);
        job_code (ajob_code);
        rate (arate);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.9.5 Facility Specification

```
/* Class Declaration */
ROSE_DECLARE (facility) : virtual public Resource {
 private:
        float PERSISTENT_square_feet_allocated;
        float PERSISTENT_cost_per_sq_ft_per_time_unit;

 public:
        ROSE_DECLARE_MEMBERS(facility);

/* Access and Update Methods */
/* square_feet_allocated Access Methods */
float square_feet_allocated()
{       return ROSE_GET_PRIM (float,PERSISTENT_square_feet_allocated);
}
void square_feet_allocated (float asquare_feet_allocated)
{       ROSE_PUT_PRIM
(float,PERSISTENT_square_feet_allocated,asquare_feet_allocated); }

/* cost_per_sq_ft_per_time_unit Access Methods */
float cost_per_sq_ft_per_time_unit()
{       return ROSE_GET_PRIM (float,PERSISTENT_cost_per_sq_ft_per_time_unit);
}
void cost_per_sq_ft_per_time_unit (float acost_per_sq_ft_per_time_unit)
{       ROSE_PUT_PRIM
(float,PERSISTENT_cost_per_sq_ft_per_time_unit,acost_per_sq_ft_per_time_unit); }
/* Constructors */
facility ();
facility (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters,
        float asquare_feet_allocated,
        float acost_per_sq_ft_per_time_unit );
};
```

/* Methods Implementation */

```
facility::facility () {
        PERSISTENT_square_feet_allocated = 0;
        PERSISTENT_cost_per_sq_ft_per_time_unit = 0;
        ROSE_CTOR_EXTENSIONS;
}
```

facility::facility (
STR aresource_name,
STR aresource_code,
ListOfParameter * aparameters,
float asquare_feet_allocated,
float acost_per_sq_ft_per_time_unit)

```
{
        resource_name (aresource_name);
        resource_code (aresource_code);
        parameters (aparameters);
        square_feet_allocated (asquare_feet_allocated);
        cost_per_sq_ft_per_time_unit (acost_per_sq_ft_per_time_unit);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.10 ComplexRule Specification

/* Class Declaration */

```
ROSE_DECLARE (ComplexRule) : virtual public RoseStructure {
 private:
        ListOfRules *  PERSISTENT_lrule;

 public:
        ROSE_DECLARE_MEMBERS(ComplexRule);
```

/* Access and Update Methods */

```
/* lrule Access Methods */
ListOfRules * lrule();
void lrule (ListOfRules *  alrule)
{       ROSE_PUT_OBJ (ListOfRules,PERSISTENT_lrule,alrule); }

/* Constructors */
ComplexRule ();
ComplexRule (
        ListOfRules *  alrule );
```

```
/* CLASS DECLARATION EXTENSIONS */
int evaluate();
};

/* Methods Implementation */
ComplexRule::ComplexRule () {
        PERSISTENT_lrule = NULL;
        ROSE_CTOR_EXTENSIONS;
}

ComplexRule::ComplexRule (
        ListOfRules *  alrule )
{
        lrule (alrule);
        ROSE_CTOR_EXTENSIONS;
}

ListOfRules * ComplexRule :: lrule()
{       if( !PERSISTENT_lrule)
        if( this->isPersistent())
                lrule (pnewIn (design()) ListOfRules);
        else    lrule (new ListOfRules);
        return ROSE_GET_OBJ (ListOfRules,PERSISTENT_lrule);
}

int ComplexRule::evaluate() {}
```

5.2.11 Rules Specification

```
/* Class Declaration */
ROSE_DECLARE (Rules) : virtual public RoseStructure {
 private:
        Expression *  PERSISTENT_exp1;
        AND_Op PERSISTENT_And1;

 public:
        ROSE_DECLARE_MEMBERS(Rules);

/* Access and Update Methods */
/* exp1 Access Methods */
Expression * exp1()
  { return ROSE_GET_OBJ (Expression,PERSISTENT_exp1); }
void exp1 (Expression * aexp1)
{       ROSE_PUT_OBJ(Expression,PERSISTENT_exp1,aexp1); }

/* And1 Access Methods */
AND_Op And1()
{       return ROSE_GET_PRIM (AND_Op,PERSISTENT_And1);
}
void And1 (AND_Op aAnd1)
{       ROSE_PUT_PRIM (AND_Op,PERSISTENT_And1,aAnd1); }

/* Constructors */
Rules ();
Rules (
        Expression * aexp1,
        AND_Op aAnd1 );
};
```

/* Methods Implementation */

```
Rules::Rules () {
        PERSISTENT_exp1 = NULL;
        PERSISTENT_And1 = (AND_Op) NULL_ENUM;
        ROSE_CTOR_EXTENSIONS;
}

Rules::Rules (
        Expression * aexp1,
        AND_Op aAnd1 )
{
        exp1 (aexp1);
        And1 (aAnd1);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.12 Expression Specification

/* Class Declaration */

```
ROSE_DECLARE (Expression) : public RoseUnion  {
 public:

ROSE_DECLARE_MEMBERS(Expression);
```

/* Access and Update Methods */

```
BOOL is_Equation()
{       return (getAttribute() == getAttribute("_Equation"));
}
Equation *  _Equation()
{       return ROSE_GET_OBJ (Equation,PERSISTENT_data.value.aPtr); }

void _Equation (Equation *  a_Equation)
{       this->putAttribute("_Equation");
        if (!ROSE.error())
                ROSE_PUT_OBJ(Equation,PERSISTENT_data.value.aPtr,a_Equation); }

BOOL is_ComplexExp()
{       return (getAttribute() == getAttribute("_ComplexExp"));
}
ComplexExp *  _ComplexExp()
{       return ROSE_GET_OBJ (ComplexExp,PERSISTENT_data.value.aPtr); }

void _ComplexExp (ComplexExp *  a_ComplexExp)
{       this->putAttribute("_ComplexExp");
        if (!ROSE.error())

        ROSE_PUT_OBJ(ComplexExp,PERSISTENT_data.value.aPtr,a_ComplexExp); }
```

```
BOOL is_SimpleExp()
{       return (getAttribute() == getAttribute("_SimpleExp"));
}
SimpleExp *  _SimpleExp()
{       return ROSE_GET_OBJ (SimpleExp,PERSISTENT_data.value.aPtr); }

void _SimpleExp (SimpleExp * a_SimpleExp)
{       this->putAttribute("_SimpleExp");
        if (!ROSE.error())
                ROSE_PUT_OBJ(SimpleExp,PERSISTENT_data.value.aPtr,a_SimpleExp); }

BOOL is_StringValue()
{       return (getAttribute() == getAttribute("_StringValue"));
}
StringValue *  _StringValue()
{       return ROSE_GET_OBJ (StringValue,PERSISTENT_data.value.aPtr); }

void _StringValue (StringValue * a_StringValue)
{       this->putAttribute("_StringValue");
        if (!ROSE.error())
                ROSE_PUT_OBJ(StringValue,PERSISTENT_data.value.aPtr,a_StringValue);
}

/* Constructor */
Expression ();
};

/* Methods Implementation */
Expression::Expression () {
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.13 ComplexExp Specification

```
/* Class Declaration */
ROSE_DECLARE (ComplexExp) : virtual public RoseStructure {
 private:
        Equation * PERSISTENT_Equ1;
        Equiv_Op PERSISTENT_EquivOp1;
        Expression * PERSISTENT_Exp1;

 public:
        ROSE_DECLARE_MEMBERS(ComplexExp);

/* Access and Update Methods */
/* Equ1 Access Methods */
Equation * Equ1()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_Equ1); }
void Equ1 (Equation * aEqu1)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_Equ1,aEqu1); }

/* EquivOp1 Access Methods */
Equiv_Op EquivOp1()
{       return ROSE_GET_PRIM (Equiv_Op,PERSISTENT_EquivOp1);
}
void EquivOp1 (Equiv_Op aEquivOp1)
{       ROSE_PUT_PRIM (Equiv_Op,PERSISTENT_EquivOp1,aEquivOp1); }

/* Exp1 Access Methods */
Expression * Exp1()
  { return ROSE_GET_OBJ (Expression,PERSISTENT_Exp1); }
void Exp1 (Expression * aExp1)
{       ROSE_PUT_OBJ(Expression,PERSISTENT_Exp1,aExp1); }

/* Constructors */
ComplexExp ();
ComplexExp (
        Equation *  aEqu1,
        Equiv_Op aEquivOp1,
        Expression *  aExp1 );
};

/* Methods Implementation */
ComplexExp::ComplexExp () {
        PERSISTENT_Equ1 = NULL;
        PERSISTENT_EquivOp1 = (Equiv_Op) NULL_ENUM;
        PERSISTENT_Exp1 = NULL;
        ROSE_CTOR_EXTENSIONS;
}

ComplexExp::ComplexExp (
        Equation *  aEqu1,
        Equiv_Op aEquivOp1,
        Expression *  aExp1 )
{
        Equ1 (aEqu1);
        EquivOp1 (aEquivOp1);
        Exp1 (aExp1);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.14 SimpleExp Specification

```
/* Class Declaration */
ROSE_DECLARE (SimpleExp) : virtual public RoseStructure {
 private:
        Unary_Op PERSISTENT_Not1;
        DataDictStr *  PERSISTENT_DataDictVar;

 public:
        ROSE_DECLARE_MEMBERS(SimpleExp);

/* Access and Update Methods */
/* Not1 Access Methods */
Unary_Op Not1()
{       return ROSE_GET_PRIM (Unary_Op,PERSISTENT_Not1);
}
void Not1 (Unary_Op aNot1)
{       ROSE_PUT_PRIM (Unary_Op,PERSISTENT_Not1,aNot1); }

/* DataDictVar Access Methods */
DataDictStr * DataDictVar()
{       return ROSE_GET_OBJ (DataDictStr,PERSISTENT_DataDictVar);
}
void DataDictVar (DataDictStr * aDataDictVar)
{       ROSE_PUT_OBJ (DataDictStr,PERSISTENT_DataDictVar,aDataDictVar); }

/* Constructors */
SimpleExp ();
SimpleExp (
        Unary_Op aNot1,
        DataDictStr *  aDataDictVar );
};

/* Methods Implementation */
SimpleExp::SimpleExp () {
        PERSISTENT_Not1 = (Unary_Op) NULL_ENUM;
        PERSISTENT_DataDictVar = NULL;
        ROSE_CTOR_EXTENSIONS;
}

SimpleExp::SimpleExp (
        Unary_Op aNot1,
        DataDictStr *  aDataDictVar )
{
        Not1 (aNot1);
        DataDictVar (aDataDictVar);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.15 Equation Specification

```
/* Class Declaration */
ROSE_DECLARE (Equation) : public RoseUnion  {
 public:

ROSE_DECLARE_MEMBERS(Equation);

/* Access and Update Methods */
BOOL is_Term()
{       return (getAttribute() == getAttribute("_Term"));
}
Term *  _Term()
{       return ROSE_GET_OBJ (Term,PERSISTENT_data.value.aPtr); }

void _Term (Term *  a_Term)
{       this->putAttribute("_Term");
        if (!ROSE.error())
        ROSE_PUT_OBJ(Term,PERSISTENT_data.value.aPtr,a_Term); }

BOOL is_ComplexEquation()
{       return (getAttribute() == getAttribute("_ComplexEquation"));
}
ComplexEquation *  _ComplexEquation()
{       return ROSE_GET_OBJ (ComplexEquation,PERSISTENT_data.value.aPtr); }

void _ComplexEquation (ComplexEquation *  a_ComplexEquation)
{       this->putAttribute("_ComplexEquation");
        if (!ROSE.error())

        ROSE_PUT_OBJ(ComplexEquation,PERSISTENT_data.value.aPtr,a_ComplexEquation); }

/* Constructor */
Equation ();

/* CLASS DECLARATION EXTENSIONS */
float evaluate()
};
```

/* Methods Implementation */
Equation::Equation () {
```
        ROSE_CTOR_EXTENSIONS;
}
```

float Equation::evaluate() {}

5.2.16 ComplexEquation Specification

/* Class Declaration */
ROSE_DECLARE (**ComplexEquation**) : virtual public RoseStructure {
```
 private:
        Term *  PERSISTENT_Var1;
        Operator PERSISTENT_Oper1;
        Equation *  PERSISTENT_Value;

 public:
        ROSE_DECLARE_MEMBERS(ComplexEquation);
```

/* Access and Update Methods */
```
/* Var1 Access Methods */
Term * Var1()
  { return ROSE_GET_OBJ (Term,PERSISTENT_Var1); }
void Var1 (Term * aVar1)
{       ROSE_PUT_OBJ(Term,PERSISTENT_Var1,aVar1); }

/* Oper1 Access Methods */
Operator Oper1()
{       return ROSE_GET_PRIM (Operator,PERSISTENT_Oper1);
}
void Oper1 (Operator aOper1)
{       ROSE_PUT_PRIM (Operator,PERSISTENT_Oper1,aOper1); }
```

```
/* Value Access Methods */
Equation * Value()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_Value); }
void Value (Equation * aValue)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_Value,aValue); }

/* Constructors */
ComplexEquation ();
ComplexEquation (
        Term *  aVar1,
        Operator aOper1,
        Equation *  aValue );
};
```

/* Methods Implementation */

```
ComplexEquation::ComplexEquation () {
        PERSISTENT_Var1 = NULL;
        PERSISTENT_Oper1 = (Operator) NULL_ENUM;
        PERSISTENT_Value = NULL;
        ROSE_CTOR_EXTENSIONS;
}

ComplexEquation::ComplexEquation (
        Term *  aVar1,
        Operator aOper1,
        Equation *  aValue )
{
        Var1 (aVar1);
        Oper1 (aOper1);
        Value (aValue);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.17 ParenEquation Specification

/* Class Declaration */

```
ROSE_DECLARE (ParenEquation) : virtual public RoseStructure {
 private:
        LParen PERSISTENT_Lparenthesis;
        Equation *  PERSISTENT_Equ;
        RParen PERSISTENT_Rparenthesis;

 public:
        ROSE_DECLARE_MEMBERS(ParenEquation);
```

/* Access and Update Methods */

```
/* Lparenthesis Access Methods */
LParen Lparenthesis()
{       return ROSE_GET_PRIM (LParen,PERSISTENT_Lparenthesis);
}
void Lparenthesis (LParen aLparenthesis)
{       ROSE_PUT_PRIM (LParen,PERSISTENT_Lparenthesis,aLparenthesis); }
```

```
/* Equu Access Methods */
Equation * Equ()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_Equ); }
void Equ (Equation * aEqu)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_Equ,aEqu); }

/* Rparenthesis Access Methods */
RParen Rparenthesis()
{       return ROSE_GET_PRIM (RParen,PERSISTENT_Rparenthesis);
}
void Rparenthesis (RParen aRparenthesis)
{       ROSE_PUT_PRIM (RParen,PERSISTENT_Rparenthesis,aRparenthesis); }

/* Constructors */
ParenEquation ();
ParenEquation (
        LParen aLparenthesis,
        Equation *  aEqu,
        RParen aRparenthesis );
};

/* Methods Implementation */
ParenEquation::ParenEquation () {
        PERSISTENT_Lparenthesis = (LParen) NULL_ENUM;
        PERSISTENT_Equ = NULL;
        PERSISTENT_Rparenthesis = (RParen) NULL_ENUM;
        ROSE_CTOR_EXTENSIONS;
}

ParenEquation::ParenEquation (
        LParen aLparenthesis,
        Equation *  aEqu,
        RParen aRparenthesis )
{
        Lparenthesis (aLparenthesis);
        Equ (aEqu);
        Rparenthesis (aRparenthesis);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.18 Term Specification

```
/* Class Declaration */
ROSE_DECLARE (Term) : public RoseUnion  {
  public:

ROSE_DECLARE_MEMBERS(Term);

/* Access and Update Methods */
BOOL is_Const()
{       return (getAttribute() == getAttribute("_Const"));
}
Const * _Const()
{       return ROSE_GET_OBJ (Const,PERSISTENT_data.value.aPtr); }

void _Const (Const * a_Const)
{       this->putAttribute("_Const");
        if (!ROSE.error())
                ROSE_PUT_OBJ(Const,PERSISTENT_data.value.aPtr,a_Const); }

BOOL is_DataDictStr()
{       return (getAttribute() == getAttribute("_DataDictStr"));
}
DataDictStr * _DataDictStr()
{       return ROSE_GET_OBJ (DataDictStr,PERSISTENT_data.value.aPtr); }

void _DataDictStr (DataDictStr * a_DataDictStr)
{       this->putAttribute("_DataDictStr");
        if (!ROSE.error())
                ROSE_PUT_OBJ(DataDictStr,PERSISTENT_data.value.aPtr,a_DataDictStr);
}

BOOL is_ParenEquation()
{       return (getAttribute() == getAttribute("_ParenEquation"));
}
ParenEquation * _ParenEquation()
{       return ROSE_GET_OBJ (ParenEquation,PERSISTENT_data.value.aPtr); }

void _ParenEquation (ParenEquation * a_ParenEquation)
{       this->putAttribute("_ParenEquation");
        if (!ROSE.error())

        ROSE_PUT_OBJ(ParenEquation,PERSISTENT_data.value.aPtr,a_ParenEquation); }

/* Constructor */
Term ();
};

/* Methods Implementation */
Term::Term () {
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.19 Const Specification

```
/* Class Declaration */
ROSE_DECLARE (Const) : public RoseUnion {
 public:

ROSE_DECLARE_MEMBERS(Const);

/* Access and Update Methods */
BOOL is_float()
{       return (getAttribute() == getAttribute("_float"));
}
float _float()
{       return (float) ROSE_GET_PRIM (float,PERSISTENT_data.value.aFloat); }

void _float (float a_float)
{       this->putAttribute("_float");
        if (!ROSE.error())
                ROSE_PUT_PRIM(float,PERSISTENT_data.value.aFloat,a_float); }

BOOL is_int()
{       return (getAttribute() == getAttribute("_int"));
}
int _int()
{       return (int) ROSE_GET_PRIM (int,PERSISTENT_data.value.anInt); }

void _int (int a_int)
{       this->putAttribute("_int");
        if (!ROSE.error())
                ROSE_PUT_PRIM(int,PERSISTENT_data.value.anInt,a_int); }

/* Constructor */
Const ();
};
```

/* Methods Implementation */
Const::Const () {

```
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.20 AND_Op Specification

/* Enumerated Type */

```
enum AND_Op {
        AND_Op_NULL = NULL_ENUM,
        AND_Op_Comma = 0
};
```

5.2.21 Operator Specification

/* Enumerated Type */

```
enum Operator {
        Operator_NULL = NULL_ENUM,
        Operator_Multiply = 0,
        Operator_Divide,
        Operator_Add,
        Operator_Subtract
};
```

5.2.22 Unary_Op Specification

/* Enumerated Type */

```
enum Unary_Op {
        Unary_Op_NULL = NULL_ENUM,
        Unary_Op_U_Op = 0
};
```

5.2.23 Equiv_Op Specification

```
/* Enumerated Type */
enum Equiv_Op {
        Equiv_Op_NULL = NULL_ENUM,
        Equiv_Op_Less = 0,
        Equiv_Op_LessEqual,
        Equiv_Op_Greater,
        Equiv_Op_GreaterEqual,
        Equiv_Op_Equal,
        Equiv_Op_NotEqual
};
```

5.2.24 StringValue Specification

```
/* Class Declaration */
ROSE_DECLARE (StringValue) : v:rtual public RoseStructure {
 private:
        DQuote PERSISTENT_quote1;
        STR PERSISTENT_value1;
        DQuote PERSISTENT_quote2;

 public:
        ROSE_DECLARE_MEMBERS(StringValue);

/* Access and Update Methods */
/* quote1 Access Methods */
DQuote quote1()
{       return ROSE_GET_PRIM (DQuote,PERSISTENT_quote1);
}
void quote1 (DQuote aquote1)
{       ROSE_PUT_PRIM (DQuote,PERSISTENT_quote1,aquote1); }

/* value1 Access Methods */
STR value1()
{       return ROSE_GET_PRIM (STR,PERSISTENT_value1);
}
void value1 (STR avalue1)
{       ROSE_PUT_PRIM (STR,PERSISTENT_value1,avalue1); }

/* quote2 Access Methods */
DQuote quote2()
{       return ROSE_GET_PRIM (DQuote,PERSISTENT_quote2);
}
void quote2 (DQuote aquote2)
{       ROSE_PUT_PRIM (DQuote,PERSISTENT_quote2,aquote2); }

/* Constructors */
StringValue ();
StringValue (
        DQuote aquote1,
        STR avalue1,
        DQuote aquote2 );
};
```

/* Methods Implementation */
```
StringValue::StringValue () {
        PERSISTENT_quote1 = (DQuote) NULL_ENUM;
        PERSISTENT_value1 = NULL;
        PERSISTENT_quote2 = (DQuote) NULL_ENUM;
        ROSE_CTOR_EXTENSIONS;
}

StringValue::StringValue (
        DQuote aquote1,
        STR avalue1,
        DQuote aquote2 )
{
        quote1 (aquote1);
        value1 (avalue1);
        quote2 (aquote2);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.25 DataDictStr Specification

/* Abstract Base Class Declaration */
```
ROSE_DECLARE (DataDictStr) : virtual public RoseStructure {
 private:

 public:
        ROSE_DECLARE_MEMBERS(DataDictStr);
```

/* Access and Update Methods */
```
/* Constructors */
DataDictStr ();
};
```

/* Methods Implementation */
```
DataDictStr::DataDictStr () {
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.25.1 EntityName Specification

/* Class Declaration */
```
ROSE_DECLARE (EntityName) : virtual public DataDictStr {
 private:
        STR PERSISTENT_name;

 public:
        ROSE_DECLARE_MEMBERS(EntityName);
/* Access and Update Methods */
/* name Access Methods */
STR name()
{       return ROSE_GET_PRIM (STR,PERSISTENT_name);
}
void name (STR aname)
{       ROSE_PUT_PRIM (STR,PERSISTENT_name,aname); }

/* Constructors */
EntityName ();
EntityName (
        STR aname );
};

/* Methods Implementation */
EntityName::EntityName () {
        PERSISTENT_name = NULL;
        ROSE_CTOR_EXTENSIONS;
}

EntityName::EntityName (
        STR aname )
{
        name (aname);
        ROSE_CTOR_EXTENSIONS;
}
```

5.2.25.2 EntityAttrName Specification

```
/* Class Declaration */
ROSE_DECLARE (EntityAttrName) : virtual public DataDictStr {
 private:
        ListOfString *  PERSISTENT_entityName;
        STR PERSISTENT_attrName;

 public:
        ROSE_DECLARE_MEMBERS(EntityAttrName);

/* Access and Update Methods */
/* entityName Access Methods */
ListOfString * entityName();
void entityName (ListOfString *  aentityName)
{       ROSE_PUT_OBJ (ListOfString,PERSISTENT_entityName,aentityName); }

/* attrName Access Methods */
STR attrName()
{       return ROSE_GET_PRIM (STR,PERSISTENT_attrName);
}
void attrName (STR aattrName)
{       ROSE_PUT_PRIM (STR,PERSISTENT_attrName,aattrName); }

/* Constructors */
EntityAttrName ();
EntityAttrName (
        ListOfString *  aentityName,
        STR aattrName );
};

/* Methods Implementation */
EntityAttrName::EntityAttrName () {
        PERSISTENT_entityName = NULL;
        PERSISTENT_attrName = NULL;
        ROSE_CTOR_EXTENSIONS;
}

EntityAttrName::EntityAttrName (
        ListOfString *  aentityName,
        STR aattrName )
{
        entityName (aentityName);
        attrName (aattrName);
        ROSE_CTOR_EXTENSIONS;
}

ListOfString * EntityAttrName :: entityName()
{       if( !PERSISTENT_entityName)
        if( this->isPersistent())
                entityName (pnewIn (design()) ListOfString);
        else    entityName (new ListOfString);
        return ROSE_GET_OBJ (ListOfString,PERSISTENT_entityName);
}
```

5.3 Analyzer

The manufacturing Analyzer is a subsystem of MO which is responsible for performing the manufacturability analysis based on what the user has selected from the user interface form. The Analyzer provides the user with the ability to perform a process selection, calculate yield and rework, and calculate time and cost. The Advisor uses the output of the Analyzer runs which it then displays to the user. Following are the object diagram, state transition diagram, and corresponding specification and methods for the Analyzer class/object.

5.3.1 Object Diagram

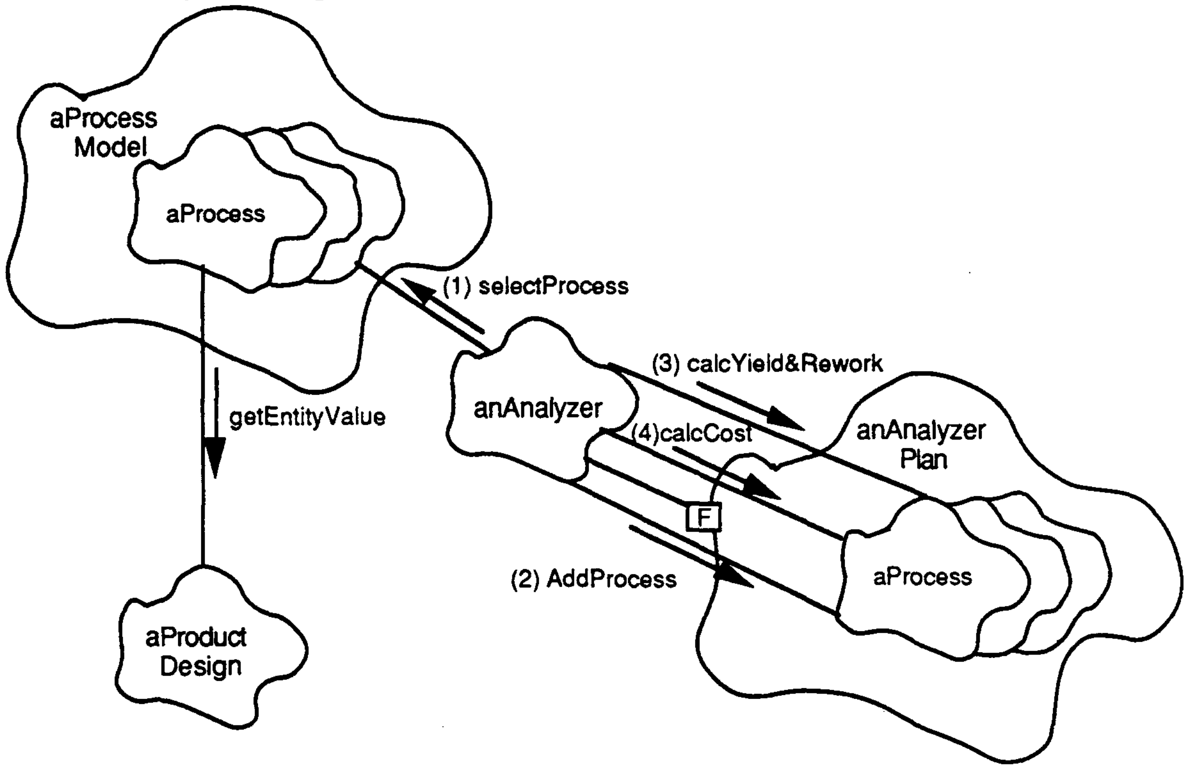


Figure 5.3-1 Analyzer Object Diagram

5.3.2 State Transition Diagram

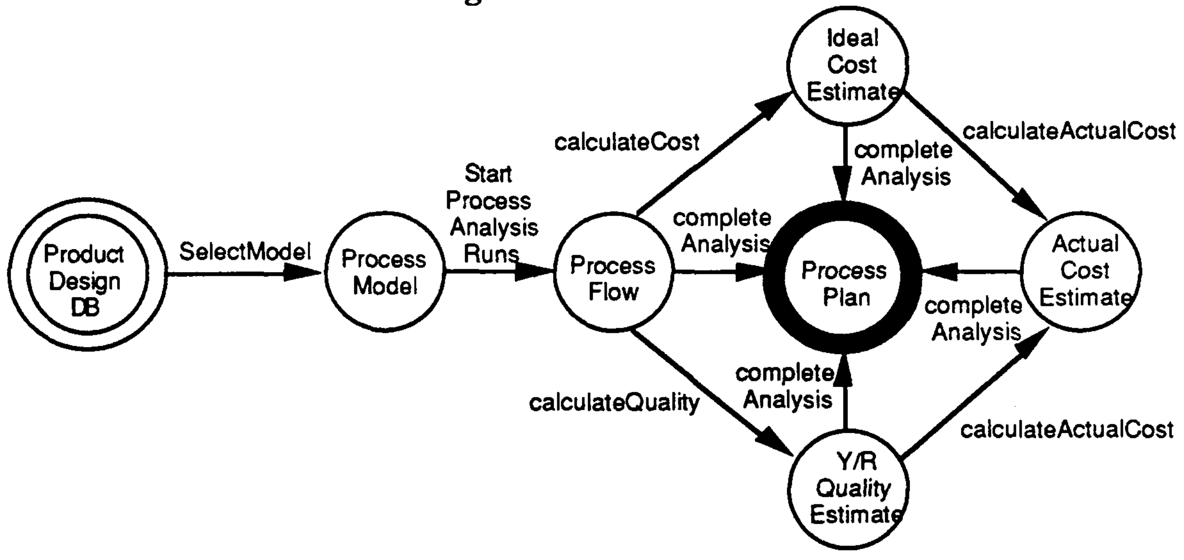


Figure 5.3-2 Analyzer State Transition Diagram

5.3.3 Analyzer Specification

```
ROSE_DECLARE (Analyzer) : virtual public RoseStructure {
 private:
        STR PERSISTENT_productDesignName;
        ProcessModel *  PERSISTENT_pModel;
        STR PERSISTENT_dateTime;
        Process *  PERSISTENT_plan;

 public:
        ROSE_DECLARE_MEMBERS(Analyzer);

/* Access and Update Methods */
/* productDesignName Access Methods */
STR productDesignName()
{       return ROSE_GET_PRIM (STR,PERSISTENT_productDesignName);
}
void productDesignName (STR aproductDesignName)
{       ROSE_PUT_PRIM (STR,PERSISTENT_productDesignName,aproductDesignName);
}

/* pModel Access Methods */
ProcessModel * pModel()
{       return ROSE_GET_OBJ (ProcessModel,PERSISTENT_pModel);
}
void pModel (ProcessModel * apModel)
{       ROSE_PUT_OBJ (ProcessModel,PERSISTENT_pModel,apModel); }

/* dateTime Access Methods */
STR dateTime()
{       return ROSE_GET_PRIM (STR,PERSISTENT_dateTime);
}
void dateTime (STR adateTime)
{       ROSE_PUT_PRIM (STR,PERSISTENT_dateTime,adateTime); }

/* plan Access Methods */
Process * plan()
{       return ROSE_GET_OBJ (Process,PERSISTENT_plan);
}
void plan (Process * aplan)
{       ROSE_PUT_OBJ (Process,PERSISTENT_plan,aplan); }

/* Constructors */
Analyzer ();
Analyzer (
        STR aproductDesignName,
        ProcessModel *  apModel,
        STR adateTime,
        Process *  aplan );

/* CLASS DECLARATION EXTENSIONS */
Process* PerformAnalysis();
};

/* Methods Implementation */
Analyzer::Analyzer () {
        PERSISTENT_productDesignName = NULL;
        PERSISTENT_pModel = NULL;
        PERSISTENT_dateTime = NULL;
        PERSISTENT_plan = NULL;
        ROSE_CTOR_EXTENSIONS;
}

Analyzer::Analyzer (
        STR aproductDesignName,
        ProcessModel * apModel,
        STR adateTime,
        Process * aplan )
{
        productDesignName (aproductDesignName);
        pModel (apModel);
        dateTime (adateTime);
        plan (aplan);
        ROSE_CTOR_EXTENSIONS;
}

Process* Analyzer::PerformAnalysis() {}
```

5.4 Advisor

The Advisor is the subsystem of MO that is responsible for displaying the results produced by each process selected during an Analyzer run. The user can select analysis runs to view. The user can display process, yield, rework, or costing results as graphs, and can also view complete analysis data to the screen or to file in report format.

The Advisor graphs will be implemented using XRT/Graph for Motif widget which displays data graphically in a window. The graph widget has resources which determine how the graph will look and behave. We will be writing methods attached to the Advisor managing object that will take the output results from the Analyzer subsystem, and display them as pictured in section 4.3 of the Advisor user interface design section.

The graph widget has resources which allow programmatic control of the following items:

- graph type (bar, stacked bar, line, and pie).
- header and footer positioning, border style, text, font, and color.
- data styles: line colors and patterns, fill color and patterns, line thickness, point style, size and color.
- legend positioning, orientation, border style, anchor, font and color.
- graph positioning, border style, color, width, height, and 3D effect.
- point and set labels.
- axis maximum and minimum, numbering increment, tick increment, grid increment, font, origin, and precision.

- window background and foreground color.
- text areas.
- double buffering.
- axis inversion.
- data transposition.
- marker positioning.

XRT/graph also provides several procedures and methods which allocate and load data structures containing the numbers to be graphed, output a representation of the graph in Postscript format, assist the developer in dealing with user-events, and assist the developer with setting and getting indexed resources.

5.5 Modeler

The process Modeler is the subsystem of MO that is responsible for capturing and modifying manufacturing process models. The Modeler provides a graphical user interface where the user can capture process dependencies to other processes, selection rules, operations, and resources. The output of the Modeler is a ProcessModel object which is comprised of a dependency graph of process nodes. The ProcessModel object is used by the Analyzer and the Advisor to select the manufacturing processes that are used in the cost, yield, and rework calculations. Following are the object diagram, state transition diagram, and corresponding specification and methods for the Modeler class/object.

5.5.1 Object Diagram

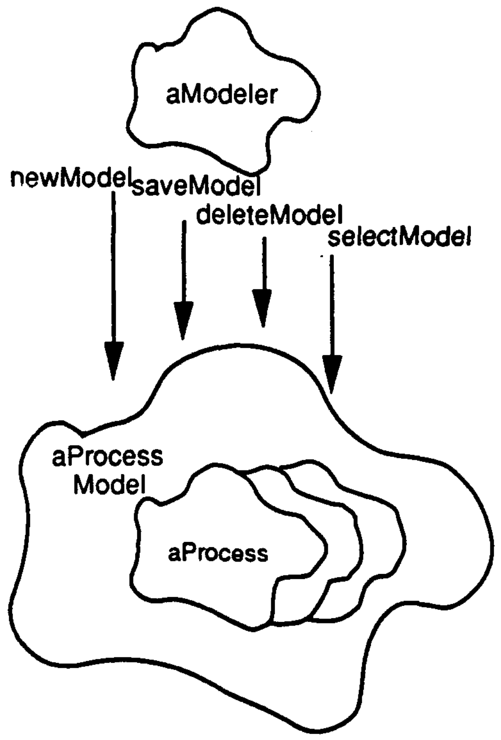


Figure 5.5-1 Modeler Object Diagram

5.5.2 State Transistion Diagram

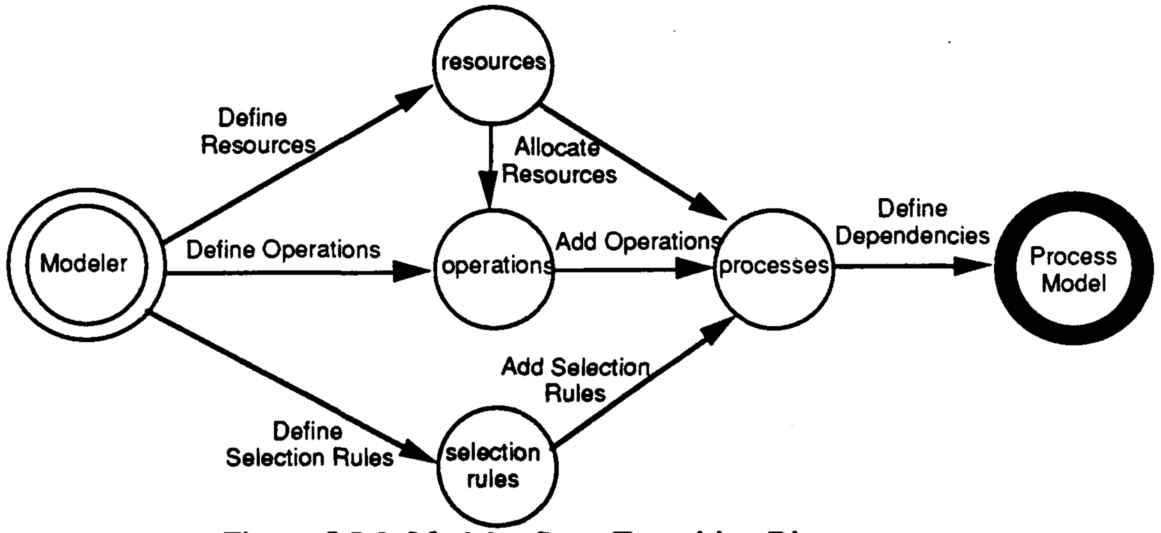


Figure 5.5-2 Modeler State Transition Diagram

5.5.3 Modeler Class Specification

```
/* Class Specification */
ROSE_DECLARE (Modeler) : virtual public RoseStructure {
 private:
        ProcessModel *  PERSISTENT_current_model;
public:
            ROSE_DECLARE_MEMBERS(Modeler);

/* Access and Update Methods */
/* current_model Access Methods */
ProcessModel * current_model()
{           return ROSE_GET_OBJ (ProcessModel,PERSISTENT_current_model);
}
void current_model (ProcessModel * acurrent_model)
{           ROSE_PUT_OBJ (ProcessModel,PERSISTENT_current_model,acurrent_model); }
/* Constructors */
Modeler ();
Modeler (
            ProcessModel *  acurrent_model );
/* CLASS DECLARATION EXTENSIONS */
ProcessModel *readModel();
void writeModel();
};

/* Methods Implementation */
ProcessModel* Modeler::readModel() {}
void Modeler::writeModel() {}
```

5.6 RequirementManager API Interface

The Product Track Requirements Manager (RM) is a system designed to manage product requirements, specifications and corporate policies to support concurrent engineering. Within the MO program, Raytheon will be using the RM to manage manufacturability and producibility guidelines and evaluate product design data for compliance with those guidelines.

Cimflex has informed Raytheon that they plan to release an Application Programming Interface (API) to the RM in the first quarter of 1993. The API will consist of a library of C programming language functions. These functions will provide the ability to populate the product and requirement structures of the SQL database which the RM is using. Functions will also be provided to evaluate and interrogate requirement satisfaction values.

A task management object called RequirementManager will be developed for interaction with RM. The RequirementManager attributes and methods will be established once CIMFLEX releases its API specification. At a minimum, an interface between the MO database (ROSE) and the RM database (SQL) will be created in order to keep product data consistent between the two systems.

6. Schema Specifications

This section defines the schemas of data to be used by MO. Schemas are defined for process model data, resource data, selection rule and equation data, and PWB product data. The schemas are defined in the modeling languages EXPRESS and EXPRESS-G.

EXPRESS is a draft International Standards Organization (ISO) language for the specification of information models. It was originally developed to enable a formal specification of the forthcoming ISO 10303 standard, familiarly known as STEP. The language is also increasingly being used in many other contexts, for example in the mechanical, electronic and petro-chemical industries, as well as in other national and international standards efforts. EXPRESS-G is a graphical subset of the EXPRESS language. The graphical nature of EXPRESS-G make it a valuable tool for understanding and analyzing information models.

6.1 Process Model Schema Specification

The Process Model provides the selection logic for the MO Manufacturing Analyzer when generating the set of process nodes to be included in the cost and yield analysis. The process model is decomposed into a graph of *process nodes*. Each *process node* consists of selection rules, dependencies, and operations. Selection rules define criteria that must be satisfied for the process node to be considered in the specific overall manufacturing process. Each process node is dependent on one or more parent nodes. For each process node, an AND/OR flag is kept that specifies if the node is dependent on any one parent node being satisfied, or all parent nodes being satisfied. For a process node to be selected for inclusion in a specific instantiation of a manufacturing process, the following two criteria must be met :

1. At least one of the nodes selection rules is satisfied.
2. All of the nodes parent processes are satisfied or the node AND/OR flag is set to OR and at least one parent node is satisfied.

At each process node there is a list of operations that are performed. Each operation is annotated with an associated yield rate, rework rate, and its usage of resources. From the list of operations the Manufacturing Analyzer will determine the aggregate cost, yield, and rework for this process node.

The EXPRESS model specified in this section was created for process model representation. Figure 6.1-1 is an EXPRESS-G representation of the same model.

6.1.1 EXPRESS Schema for Process Model

This EXPRESS schema listing defines the process model. The process model schema includes two auxiliary schemas, the selection_rules schema and the resource_schema. The specification of these two schemas will follow.

EXPRESS Specification :

```
*)
INCLUDE 'rules.exp';
INCLUDE 'resource.exp';

SCHEMA process_model;

REFERENCE FROM selection_rules;
REFERENCE FROM resource_schema;

(*
```

6.1.1.1 ProcessModel Entity

A ProcessModel entity is the specification of a manufacturing process model that contains a dependency graph of Process entities. Additional data about the model is also stored including its name, author, creation date, and last modification date.

EXPRESS Specification :

```
ENTITY ProcessModel;
   name : STRING;                        -- Process Model name
   creationDate : Date;                  -- Model creation date
   modifyDate   : Date;                  -- Model last modify date
   author : STRING;                      -- Model author
   topProcess : Process;                 -- Top process in
END_ENTITY;                              -- dependency graph
```

Attribute definitions:

name: Name of the manufacturing process model.

creationDate: The date that the model was created.

modifyDate: The date that the model was last modified.

author: The author of the model.

topProcess: The root or top most process in the process model dependency graph.

6.1.1.2 Process Entity

A Process entity is the specification of a manufacturing process that contains selection rules, operations, and resources. Rules and dependencies of the manufacturing process are modeled in the process nodes. If the rules and dependencies are satisfied, then the process node is included in the overall process analysis. Processes are organized as a directed dependency graph. The dependency graph takes the form of a tree where each node can have one or more parents and one or more children. Each Process will also have a reference to its immediate siblings (i.e. its neighboring nodes at the same level in the tree).

EXPRESS Specification :

```
ENTITY Process;
    name: STRING;                              -- Process Name
    andParents : BOOLEAN                       -- AND/OR dependency flag
    rules: LIST [0:?] OF ComplexRule;          -- List of Selection Rules
    elimrules: LIST [0:?] OF ComplexRule;      -- List of Selection Rules
    resources: LIST [0:?] OF Resource;         -- List Of Resources
    parents : LIST [0:?] OF Process;           -- List of Parents (Ancestors)
    children : LIST [0:?] OF Process;          -- List of Children (Descendants)
    lsibling : OPTIONAL Process;               -- Left Sibling
    rsibling : OPTIONAL Process;               -- Right Sibling
    OperList : LIST [0:?] OF Operation;        -- List of Process Operations
END_ENTITY;
```

Attribute definitions:

name: Name of the manufacturing process.

andParents: AND/OR dependency flag. If set to TRUE, the parent nodes to which this node must be satisfied for this node to be considered for selection. If set to FALSE, a minimum of one of the parent nodes must be satisfied for this node to be considered for selection.

rules: List of selection rules. The rules are comprised of design feature entity and attributes being present or of specific values.

elimrules: List of exception rules. These rules are identical to the selection rules in their syntax. If an exception rule is satisfied then this process will not be considered for selection.

resources: List of resources used by the process node as an entity. This list of resources are associated with the process node. A separate list of resources is kept with each operation. Therefore, this list should not contain any resources that have been attached to an operation.

parents: List of processes that this node is dependent on. The nodes in this list are the immediate ancestor to this node. For this node to be selected, depending on the andParents flag, all or one of the nodes in this list must be satisfied.

children: List of processes that are dependent on this node. The nodes in this are direct descendants of this node.

rsibling: The process to the immediate right of this node on the same level of the dependency graph.

lsibling: The process to the immediate left of this node on the same level of the dependency graph.

OperList: List of operations attached to this process. If this node is selected, then the operations in this list will be evaluated for cost, yield and rework.

6.1.1.3 Operation Entity

Attached to each process is a list of operations. When a process is selected and included in the overall manufacturing process, its list of operations is evaluated for cost, rework and yield analysis. Each operation is comprised of the following: A list of resources that are required to perform the operation, the time required to setup for and actually run the operation, an efficiency rate is kept with each operation this provides a factor that when applied to the labor standard for the operation, will calculate the actual time for the operation. Data for computing scrap and rework rates can be stored in a set of lookup tables or specified by a set of rules.

EXPRESS Specification :

```
*)
        ENTITY Operation;
            name : STRING;                                  -- Operation Name
            desc: STRING;                                   -- Description
            resources: LIST [0:?] OF ResourceUtilization;   -- List Of Resources
            scrap_rate : LIST [0:?] OF Scrap;               -- Scrap rates
            rework_rate : LIST [0:?] OF Rework;             -- Rework rates
            cost : OPTIONAL OpCost;                         -- Operation Cost (for Analyzer)
        END_ENTITY;
(*
```

Attribute definitions:

name: Alpha-numeric name of the operation.

desc: Textual description of the operation.

resources: List of resources required to perform the operation.

scrap_rate: A list of a list table entries providing an indexed lookup of scrap rates based on values of entities and their attributes or an equation that when evaluated will provide the scrap rate for the operation.

rework_rate: A list of a list table entries providing an indexed lookup of rework rates based on values of entities and their attributes or an equation that when evaluated will provide the rework rate for the operation.

6.1.1.4 Scrap Entity

The scrap entity is used to represent scrap rate data. Scrap being the percentage of product parts that must be scrapped due to this operation. Scrap data is maintained in a list of scrap entities. In each entity there is a scrap rule and a corresponding scrap rate. If the scrap rule is satisfied, then the corresponding scrap rate is computed.

EXPRESS Specification :

```
*)
    ENTITY Scrap;
        scrapRule : ComplexExp;                    -- Rule to be evaluated
        scrapRate : Equation;                      -- Scrap that applies if rule is satisfied
    END_ENTITY;

(*
```

Attribute definitions:

scrapRule: The scrap rule to be evaluated.

scrapRate: The scrap rate equation to apply if the scrapRule is satisfied.

6.1.1.5 Rework Entity

The rework entity is used to represent rework rate data. Rework being the percentage of product parts that must be reworked due to this operation. Rework data is maintained in a list of rework entities. In each entity there is a rework rule and a corresponding rework rate. If the rework rule is satisfied, then the corresponding rework rate is computed. There is a list of resources associated with the rework which is used to calculate the cost of performing the rework operation.

EXPRESS Specification :

```
*)
    ENTITY Rework;
        reworkRule : ComplexExp;                   -- Rule to be evaluated
        reworkRate : Equation;                     -- Rework that applies if rule is satisfied
        resources : LIST [0:?] OF ResourceUtilization;  -- Rework resources
    END_ENTITY;

(*
```

Attribute definitions:

reworkRule: The rework rule to be evaluated.

reworkRate: The rework rate equation to apply if the reworkRule is satisfied.

resources: The resources associated with the rework.

6.1.1.6 OpCost Data

The OpCost data types and entities are used to represent calculated analyzer data associated with an operation.

EXPRESS Specification :

```
*)

    ENTITY OpCost;
        setupTime: REAL;                    -- Operation Setup Time
        runTime:  REAL;                     -- Operation Run Time
        scrapPercentage: REAL;              -- Actual Calculated Operation Scrap
        reworkPercentage: REAL;             -- Calculated Operation Rework
        reworkCost: REAL;                   -- Calculated Rework Cost
        prodQty: INTEGER;                   -- Actual Prod. QTY Required
        IdealFait: REAL;                    -- Calculate Ideal FAIT
        ActualFait: REAL;                   -- Calculate Actual Estimated FAIT
    END_ENTITY;

(*
```

Attribute definitions:

setupTime: Operation calculated setup time.

runTime: Operation calculated run time.

scrapPercentage: Operation calculated scrap percentage.

reworkPercentage: Operation calculated rework percentage.

reworkCost: Operation calculated rework cost.

prodQty: Required operation production quantity.

IdealFait: Ideal Operation Fabrication, Assembly, Inspection, and Test Cost

ActualFait: Actual Estimated Operation Fabrication, Assembly, Inspection, and Test Cost

6.1.2 EXPRESS-G Schema for Process Model

The following EXPRESS-G model (figure 6.1-1) represents the Process Model schema:

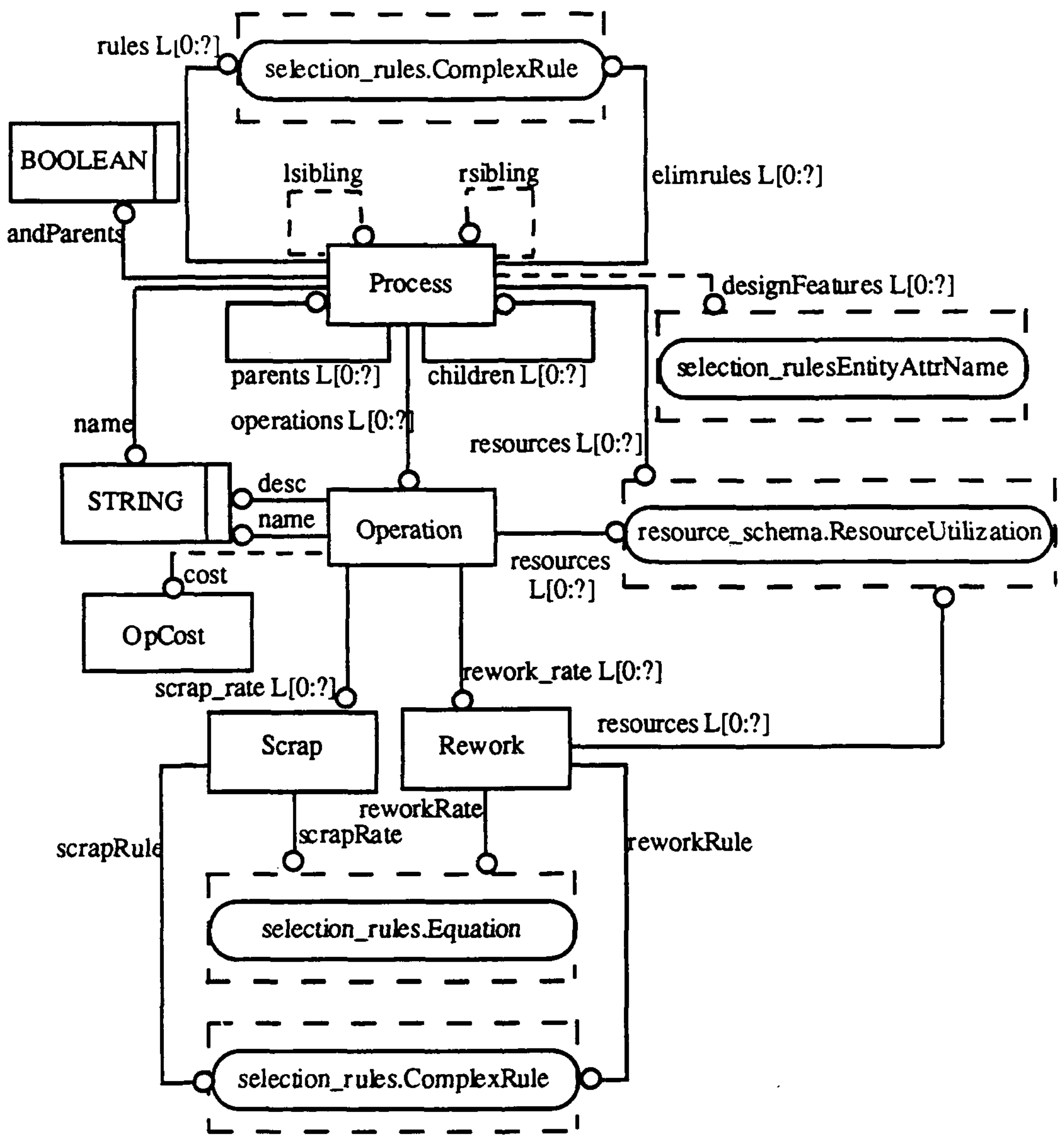


Figure 6.1-1 EXPRESS-G Model of Process Model Schema

6.1.3 EXPRESS Schema for Resource

The resource schema defines a collection of entities that are used to specify resources. A resource is any facility, labor, equipment, or consumable material used in the manufacturing process. A consumable material is a material that is used to aid the manufacturing process and is not considered raw material of the product. As defined in the schema a resource is a generic entity. Specific subtypes of the resource entity are defined to represent facilities, people, equipment, and consumable materials.

EXPRESS Specification :

```
*)
SCHEMA resource_schema;
```

6.1.3.1 ResourceUtilization Entity

The ResourceUtilization Entity is used to store which resource(s) are utilized by a process or operation.

EXPRESS Specification :

```
*)
    ENTITY ResourceUtilization;
       resource : Resource;                    -- Resource utilized
       setupTime: Equation;                    -- Setup Equation
       runTime:  Equation;                     -- RunTime Equation
       effRate : OPTIONAL REAL;                -- Efficiency Rate
    END_ENTITY;

(*
```

Attribute definitions:

resource: The resource being utilized.

setupTime: The amount of setup time required for the resource.

runTime: The amount of time that the resource is being used while running the operation.

effRate: This optional attribute provides an efficiency rate factor that when applied to a labor standard associated with an operation will provide the actual time for the operation.

6.1.3.2 Resource Entity

This is the generic resource entity. Each resource is named and can be coded of a certain type. A list of generic attributes can be attached to each resource using the parameter entity.

EXPRESS Specification :

```
*)
ENTITY Resource;
   resource_name : STRING;                     -- Resource Name
   resource_code : STRING;                     -- Resource Code
   parameters : LIST [0:?] of parameter;       -- Resource Parameters
END_ENTITY;

(*
```

Attribute definitions:

resource_name: The name string associated with the resource.

resource_code: A string used to assign a code to the resource.

parameters: A list of generic attributes that can be attached to this resource.

6.1.3.3 Parameter Entity

The parameter entity is used to define a generic attribute.

EXPRESS Specification :

```
*)

ENTITY Parameter;
    p_name : STRING;                          -- Parameter Name
    p_value : STRING;                         -- Parameter Value
END_ENTITY;
(*
```

Attribute definitions:

p_name: The name of the parameter.

p_value: The value of the parameter.

6.1.3.4 Labor Entity

The entities in this section define the labor resource. The labor entity is a subtype of the generic resource entity.

EXPRESS Specification :

```
*)

ENTITY Labor SUBTYPE OF (Resource);
    job_code : STRING;                        -- Labor Job Code
    rate : REAL;                              -- Labor Rate
END_ENTITY;

(*
```

Attribute definitions:

job_code : A unique identifier associated with the labor.

rate : The labor rate.

6.1.3.5 Equipment Entity

The equipment entity is a subtype of the generic resource entity. It is used to specify the cost of operating the equipment resource during an operation or process.

EXPRESS Specification :

```
*)

ENTITY Equipment SUBTYPE OF (Resource);
    equipment_category : STRING;              -- Equipment Category
```

```
   cost_per_time_unit : REAL;                            -- Cost Per Time Unit
END_ENTITY;
```

(*

Attribute definitions:

equipment_category: The equipment code or category.

cost_per_time_unit: The cost of operating the equipment resource per unit of time.

6.1.3.6 Facility Entity

The facility entity is a subtype of the generic resource entity. It is used to specify the cost of using the facility resource during an operation or process.

EXPRESS Specification :

```
*)
ENTITY facility SUBTYPE OF (Resource);
   square_feet_allocated : REAL;                         -- Square Feet Allocated
   cost_per_sq_ft_per_time_unit : REAL;                  -- Cost Per Sq Foot Per Time Unit
END_ENTITY;
(*
```

Attribute definitions:

square_feet_allocated: The square feet allocated to this particular operation or process.

cost_per_sq_ft_per_time_unit: The cost per square foot per time unit.

6.1.3.7 ConsumableMaterial Entity

The consumable material entity is a subtype of the generic resource entity. Consumable materials are those materials used to aid in the manufacturing of a product that are consumed by the process. These materials are not considered as part of the raw materials used in the manufacture of the product. They only aid in the production process and are consumed at some measurable rate during the process.

EXPRESS Specification :

```
*)
ENTITY ConsumableMaterial SUBTYPE OF (Resource);
   cost_per_unit : REAL;                                 -- Cost Per Unit
   resourceRates: LIST [0:?] OF ResourceConsumable;      -- list of resource rates
END_ENTITY;

ENTITY ResourceConsumable;
   aresource : Resource;                                 -- Associated Resource
   units_exhausted_per_time_unit : REAL;                 -- Units Exhausted Per Hour
END_ENTITY;
(*
```

Attribute definitions:

cost_per_unit: The cost of one unit of the consumable material.

resourceRates: The list of resource rates.

aResource: The associated Consumable Resource.

units_exhausted_per_time_unit: Units consumed per unit of time during or by the operation or process.

6.1.4 EXPRESS-G Schema for Resource

The following EXPRESS-G schema (figure 6.1-2) represents the Resource schema:

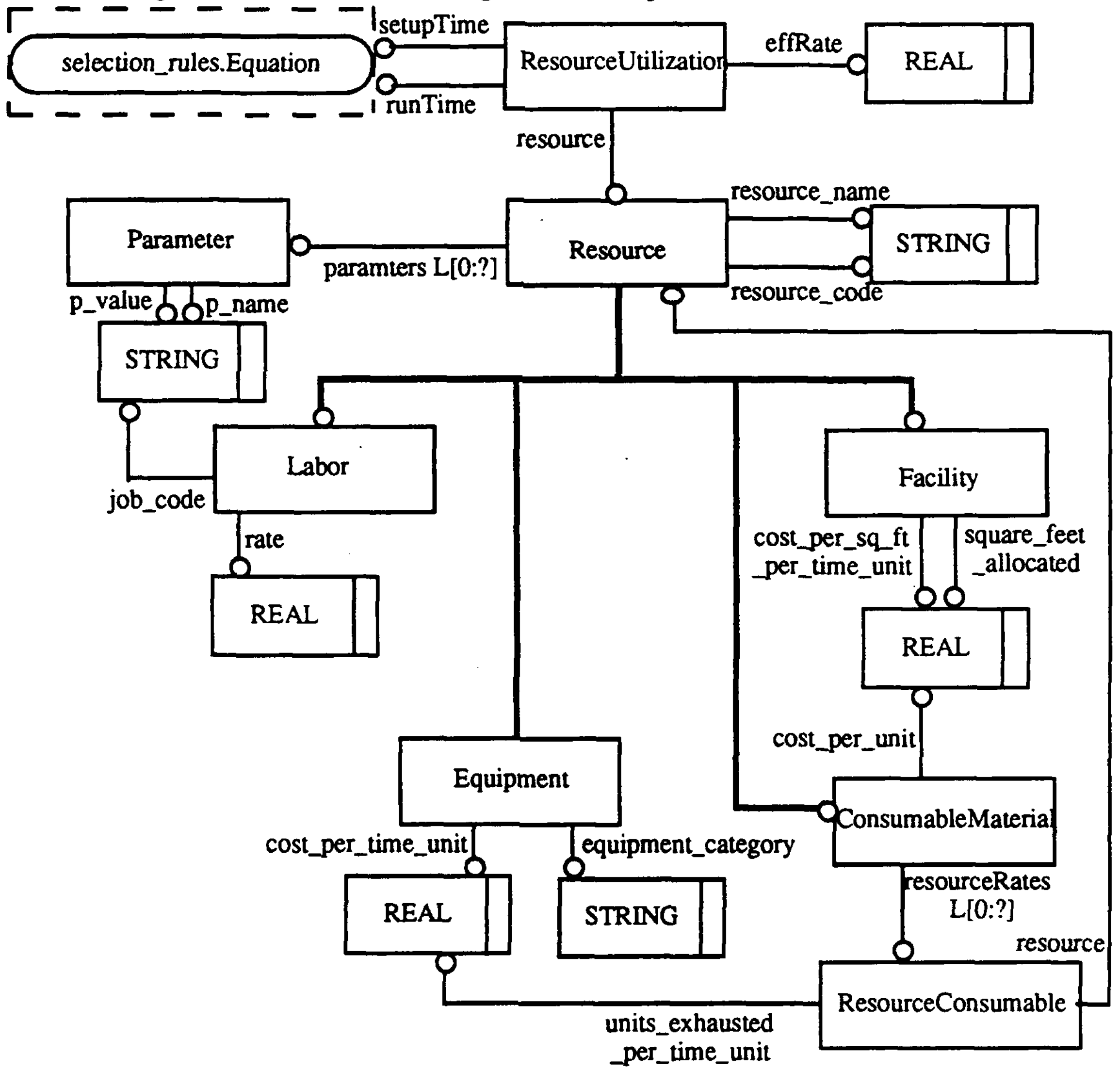


Figure 6.1-2 EXPRESS-G Model of Resources Schema

6.1.5 EXPRESS Schema for Selection Rules

This schema defines a grammar format which rules for selection and equations for evaluation can be specified. Rules are tied to process nodes and equations are tied to such entities as scrap and rework formulas. Provided below is the complete BNF (Backus-Naur Form) grammar format for the selection rules and equations which the EXPRESS schema is based on.

Complex Rule Grammar Format

```
<complexRule> := <rules> [<complexRule>]

<rules> := <expression> | <expression> , <rules>

<expression> := <equation> | <equation> <equiv_op> <expression> | <unary_op> <var> | "string"

<equation> := <term> | <term> <op> <equation>

<term> :=  <const> | <var> | ( <equation> )

<var> := <data dictionary strings>

<data dictionary strings> := <alpha> [ <alpha> | <digit> | <underscore> ]

<const> := real numbers | integers

<op> :=
        *  multiplications
        /  division
        +  addition
        -  subtraction

<unary_op> :=
        !  not

<equiv_op> :=
        <  less than
        <= less than equal to
        >  greater than
        >= greater than equal to
        =  equal to
        != not equal to
```

Operator Precedence (ordered by most --> least priority)

1	!	logical negation	
2	*	multiplications	
	/	division	(left to right)
3	+	addition	
	-	subtraction	(left to right)

4	<	less than	
	<=	less than equal to	
	>	greater than	(left to right)
	>=	greater than equal to	
5	=	equal to	
	!=	not equal to	(left to right)
6	,	AND	

6.1.5.1 Constants and Types for Rule Construction

The following is a listing of the EXPRESS source that defines symbolic constants and aggregate types that are necessary for the specification of the rules BNF:

EXPRESS Specification :

```
*)
SCHEMA selection_rules;

    CONSTANT
       Multiply                  : STRING := '*';
       Divide                    : STRING := '/';
       Add                       : STRING := '+';
       Subtract                  : STRING := '-';
       U_Op                      : STRING := '!';
       Less                      : STRING := '<';
       LessEqual                 : STRING := '<=';
       Greater                   : STRING := '>';
       GreaterEqual              : STRING := '>=';
       Equal                     : STRING := '=';
       NotEqual                  : STRING := '!=';
       LP                        : STRING := '(';
       RP                        : STRING := ')';
       Comma             : STRING := ',';
       DQ                        : STRING := '"';
    END_CONSTANT;

    TYPE DQuote = ENUMERATION OF (DQ);
    END_TYPE;

    TYPE AND_Op = ENUMERATION OF (Comma);
    END_TYPE;

    TYPE LParen = ENUMERATION OF (LP);
    END_TYPE;

    TYPE RParen = ENUMERATION OF (RP);
    END_TYPE;

    TYPE Unary_Op = ENUMERATION OF (U_Op);
    END_TYPE;

    TYPE Strings = STRING;
    END_TYPE;

    TYPE Real_numbers = REAL;
    END_TYPE;
    TYPE Integers = INTEGER;
    END_TYPE;

    TYPE
       TokenReturnValue = SELECT (Real_numbers, Integers, Strings);
    END_TYPE;

    TYPE
       Const = SELECT (Real_numbers, Integers);
    END_TYPE;

    TYPE Operator = ENUMERATION OF
       (Multiply, Divide, Add, Subtract);
    END_TYPE;

    TYPE Equiv_Op = ENUMERATION OF
       (Less, LessEqual, Greater, GreaterEqual, Equal, NotEqual);
    END_TYPE;

(*
```

6.1.5.2 DataDictStr Entity

The DataDictStr entity is an abstract base class from which two subclass have been created. The first is the EntityName class which holds the name of an entity name. The other is the EntityAttrName which is used to support the following entity attribute specification :

```
entity[.attr[.attr[... .attr]]]
```

An example of an instance of this might be :

```
line.point1.x
```

EXPRESS Specification :

```
*)

    ENTITY DataDictStr; -- abstract base class
    END_ENTITY;

    ENTITY EntityName
       SUBTYPE OF (DataDictStr);
          name : STRING;
    END_ENTITY;

(*
```

Attribute definitions:

name: The name of the entity as it appears in the product data EXPRESS model.
*)

EXPRESS Specification :

```
*)

    ENTITY EntityAttrName
```

```
        SUBTYPE OF (DataDictStr);
            entityName : STRING;
            attrName   : LIST [1:?] OF STRING;
    END_ENTITY;
```

(*

Attribute definitions:

entityName: The name of the entity as it appears in the product data EXPRESS model.

attrName: List of attribute names that correspond to the structure : .attr[.attr[... .attr]]. These attribute name should be specified as they appear in the product data EXPRESS model.

6.1.5.3 ComplexRule Entities

A complex rule is composed of a list of rules. A rule is an Expressions anded together. The following BNF segment defines the grammar of the EXPRESS entities :

<Rules> := <Expression> <AND_OP> | <Expression> <AND_OP> <Rules>

EXPRESS Specification :
*)

```
    ENTITY Rules;
        exp1 : Expression;
        And1 : And_Op;
    END_ENTITY;

    ENTITY ComplexRule;
        lrule: LIST [0:?] OF Rules;
    END_ENTITY;
```

(*

6.1.5.4 Expression Entities

The Expression syntax is represented by the following BNF segment :

<Expression> := <Equation> | <ComplexExp> | <SimpleExp> | <StringValue>

EXPRESS Specification :
*)

```
    TYPE
        Expression = SELECT (Equation, ComplexExp, SimpleExp, StringValue);
    END_TYPE;

    ENTITY StringValue;
        quote1 : DQuote;
        value1 : STRING;
        quote2 : DQuote;
    END_ENTITY;
    ENTITY ComplexExp;
        Equ1      : Equation;
        EquivOp1 : Equiv_Op;
        Exp1      : Expression;
    END_ENTITY;

    ENTITY SimpleExp;
        Not1 : Unary_Op;
        DataDictVar : DataDictStr;
    END_ENTITY;

(*
```

6.1.5.5 Equation Entities

The Equation syntax is represented by the following BNF segment :

<Equation> := <Term> | <ComplexEquation>

<u>EXPRESS Specification :</u>

```
*)

    TYPE
        Equation = SELECT (Term, ComplexEquation);
    END_TYPE;

    ENTITY ComplexEquation;
        Var1      : Term;
        Oper1     : Operator;
        Value     : Equation;
    END_ENTITY;

    ENTITY ParenEquation;
        Lparenthesis       : LParen;
        Equ                : Equation;
        Rparenthesis       : RParen;
    END_ENTITY;

(*
```

6.1.5.6 Term Entities

The Term syntax is represented by the following BNF segment :

<Term> := <Const> | <DataDictStr> | <Equation>

<u>EXPRESS Specification :</u>

```
*)

    TYPE
        Term = SELECT (Const, DataDictStr, Equation);
    END_TYPE;

END_SCHEMA;

(*
```

6.1.6 EXPRESS-G Schema for Selection Rules

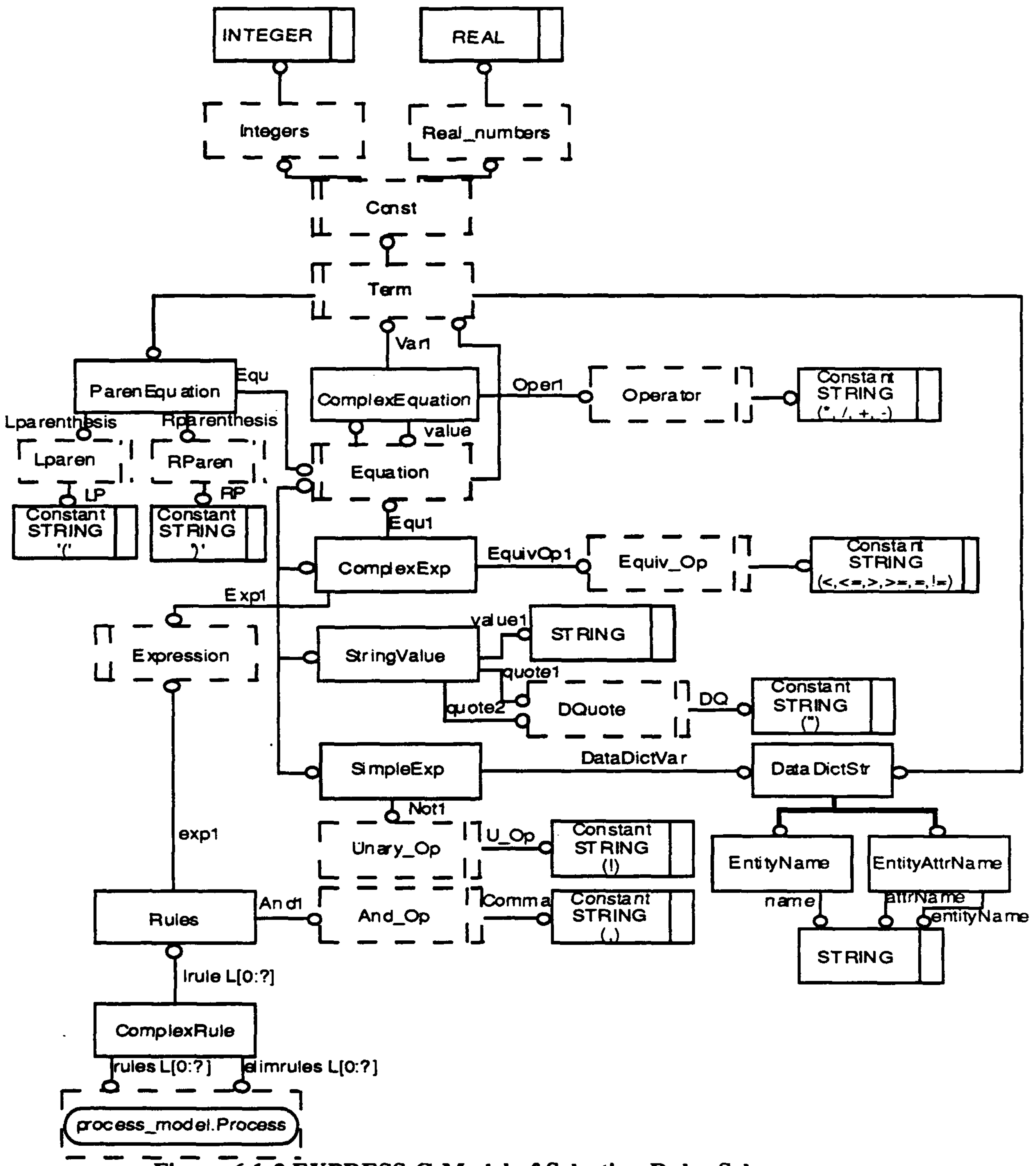


Figure 6.1-3 EXPRESS-G Model of Selection Rules Schema

6.2 Product Model Schema Specification

Product data interpretable by the MO system must be modeled in the EXPRESS language and stored as STEP objects in a repository that is interfaced to the STEP Data Access Interface (SDAI). Currently the SDAI only supports a STEP physical file. In the following sections an EXPRESS schema for a PWB product is presented. This schema was created to demonstrate the functionality of the MO system. The schema defines lists of entities that model features of a PWB.

6.2.1 Printed Wiring Board Product Data Model

At Raytheon, PWB product data is stored in the RAPIDS (Raytheon's Automated Placement and Interconnect Design System) database. Two interfaces were developed to support the transition of PWB product data to and from STEP physical files.

Generating the STEP physical file is facilitated by the interface *RAPIDS to STEP* which maps RAPIDS data items into instantiated STEP entities. We created an information model using the EXPRESS information modeling language. The model was based on the RAPIDS database. The EXPRESS information model was compiled using the STEP Tools *express2c++* compiler which generated a STEP schema and a C++ class library. The class library consists of methods for creating and referencing persistent instances of the STEP entities which are stored in a ROSE database. The STEP schema is used by the STEP Tools *STEP filer* for reading and writing the STEP physical file.

The MO system will use the STEP data directly, as well as for information exchange between the various members of the design team. At Raytheon, the top level team would most likely be using RAPIDS. This is not a requirement for using the core of the MO system. The only requirement is that the top level team and the lower level teams are capable of creating, exchanging and using the STEP physical file.

The Manufacturing Team passes back a consolidated design position to the top level. To aid in the generation of a consolidated position, conflict resolution and design merging must be supported. This is done using the STEP Toolkit from STEP Tools Inc. The *diff* tool reads two versions of a design and creates a delta file. The *difference report generator* reads the difference file and the original design, and presents each STEP entity and its attributes with the original values and its change state clearly marked with an asterisks.

Once the conflicts of the Manufacturing team members have been resolved, design versions are merged using the STEP Tools *sed* tool. The *sed* tool read the delta file created by the *diff* tool and updates the original design version. This updated version of the design will be transferred back to the top-level product team as the Manufacturing Team's consolidated position.

6.2.1.1 PWB Design Schema

This is the top level schema for the Raytheon PWB EXPRESS model. The model is primarily derived from the Raytheon's Automated Placement and Interconnect Design System (RAPIDS) data dictionary. RAPIDS is a concurrent engineering design station for Printed Wiring Boards. Its database was designed to capture data from many diverse CAE, CAD, CAM, CAT systems as well as analysis systems for thermal, reliability, critical signal analysis, and manufacturability. Emphasis was placed on making the model extremely modular and flexible.

<u>EXPRESS Specification :</u>

```
*)

INCLUDE 'rpdtypes.exp';
INCLUDE 'rpd_header.exp';
INCLUDE 'alias.exp';
INCLUDE 'annotation.exp';
INCLUDE 'cari.exp';
INCLUDE 'class.exp';
INCLUDE 'comment.exp';
INCLUDE 'dr_block.exp';
INCLUDE 'gate.exp';
INCLUDE 'net.exp';
INCLUDE 'metal_area.exp';
INCLUDE 'part.exp';
INCLUDE 'pin.exp';
INCLUDE 'route.exp';
INCLUDE 'via.exp';
INCLUDE 'xref.exp';
INCLUDE 'shape.exp';
INCLUDE 'stackup.exp';
INCLUDE 'model.exp';

SCHEMA rpd_design;

REFERENCE FROM rpdtypes_schema;
REFERENCE FROM rpd_header_schema;
REFERENCE FROM alias_schema;
REFERENCE FROM annotation_schema;
REFERENCE FROM cari_schema;
REFERENCE FROM class_schema;
REFERENCE FROM comment_schema;
REFERENCE FROM dr_block_schema;
REFERENCE FROM gate_schema;
REFERENCE FROM net_schema;
REFERENCE FROM metal_area_schema;
REFERENCE FROM part_schema;
REFERENCE FROM pin_schema;
REFERENCE FROM route_schema;
REFERENCE FROM via_schema;
REFERENCE FROM xref_schema;
REFERENCE FROM model_schema;
REFERENCE FROM shape_schema;
REFERENCE FROM stackup_schema;

ENTITY rpd_design_rec;
  alias_header : header_rec;
  aliases : LIST [0:?] of alias_rec;          -- list of aliases
  annotation_header : header_rec;
  annotations : LIST [0:?] of annotation_rec; -- list of anncotations
  cari_header : header_rec;
  cari_rules : LIST [0:?] of cari_rule_rec;   -- list of cari rules
  class_header : header_rec;
  classes : LIST [0:?] of class_rec;          -- list of classes
  comment_header : header_rec;
  comments : LIST [0:?] of comment_rec;       -- list of design comments
  dr_block_header : header_rec;
  dr_blocks : LIST [0:?] of dr_block_rec;     -- list of design rule blocks
  gate_header : header_rec;
  gates : LIST [0:?] of gate_rec;             -- list of gates
  net_header : header_rec;
  nets : LIST [0:?] of net_rec;               -- list of nets
  part_header : header_rec;
  parts : LIST [0:?] of part_rec;             -- list of parts
  pins_header : header_rec;
  pins : LIST [0:?] of pin_rec;               -- list of pins
  route_header : header_rec;
  routes : LIST [0:?] of route_rec;           -- list of routes
  vias_header : header_rec;
  vias : LIST [0:?] of via_rec;               -- list of vias
  xref_header : header_rec;
  xrefs : LIST [0:?] of xref_rec;             -- list of xrefs
  shapes_header : header_rec;
  shapes : LIST [0:?] of pad_shape_rec;       -- list of pad shapes
  stackups_header : header_rec;
  stackups : LIST [0:?] of stackup_rec;       -- list of pad stackups
  models : LIST [0:?] of model_rec;           -- list of part mechanical
models
END_ENTITY;

END_SCHEMA;
```

6.2.1.2 PWB Generic Types and Entities

This schema defines types and entities that are used throughout the entire PWB model. these types and entities are generic and low level and are used as resources by higher level entities.

<u>EXPRESS Specification :</u>
*)

```
SCHEMA rpdtypes_schema;

TYPE token = STRING; END_TYPE;

TYPE name_type = STRING; END_TYPE;

TYPE layer_type = STRING; END_TYPE;
TYPE keyword = STRING; END_TYPE;

TYPE dimension = INTEGER; END_TYPE;

TYPE shape_type = STRING; END_TYPE;

TYPE loading_type = REAL; END_TYPE;

TYPE blocking_type = STRING; END_TYPE;

-- BINARY data type is not currently supported by the EXPRESS compiler
-- Assumming 8 bit characters (256 layers, 1 bit per layer)
   TYPE bitmask = ARRAY [0:31] of STRING(1); END_TYPE;

ENTITY time_rec;
  high : INTEGER;
  low  : INTEGER;
END_ENTITY;

ENTITY r_range_rec;
  minimum : REAL;
  maximum : REAL;
END_ENTITY;

ENTITY i_range_rec;
  minimum : INTEGER;
  maximum : INTEGER;
END_ENTITY;

ENTITY r_span_rec;
  minimum : REAL;
  maximum : REAL;
  span : REAL;
END_ENTITY;

ENTITY i_span_rec;
  minimum : INTEGER;
  maximum : INTEGER;
  span : INTEGER;
END_ENTITY;

ENTITY pin_name_rec;
    device : name_type;
    gate   : name_type;
    pin    : name_type;
END_ENTITY;

ENTITY vertex_rec;
    x : dimension;
    y : dimension;
    radius : dimension;
END_ENTITY;

ENTITY point_rec;
    x : dimension;
    y : dimension;
END_ENTITY;

ENTITY loading_rec;
    rated : REAL;
    derated : REAL;
    actual : REAL;
END_ENTITY;
```

```
ENTITY attribute_rec;
    key : keyword;
    value : STRING;
END_ENTITY;

END_SCHEMA;
```

6.2.1.3 Header Data Schema

This schema defines entities for the unit and scale of other entity instances and the creation, access, and modification time entities.

EXPRESS Specification :
*)

```
SCHEMA rpd_header_schema;

REFERENCE FROM rpdtypes_schema;

ENTITY version_rec;
  name : NAME_TYPE;
  revision : NAME_TYPE;
END_ENTITY;

ENTITY header_rec;
  file_name : NAME_TYPE;
  version : NAME_TYPE;
  creation : TIME_REC;
  access : TIME_REC;
  modification : TIME_REC;
  unit : NAME_TYPE;
  scale : REAL;
  tool : NAME_TYPE;
  tool_ver : INTEGER;
  tool_rev : INTEGER;
  assembly : version_rec;
  drawing : version_rec;
  codeid : NAME_TYPE;                        -- Wire Wrap code id
  comment : STRING;
  attribute : LIST OF ATTRIBUTE_REC;
END_ENTITY;

END_SCHEMA;
```

6.2.1.4 Alias Data Schema

This is the EXPRESS schema for storing data aliases required by limitations of some CAx system (e.g. NET names in one system are restricted to a particular length that has been violated by a system that is upstream in the design process)

EXPRESS Specification :
*)

```
SCHEMA alias_schema;
```

```
REFERENCE FROM rpdtypes_schema;

ENTITY alias_list_rec;
  rapids_name : NAME_TYPE;
  alias_name : NAME_TYPE;
  object_name : NAME_TYPE;
END_ENTITY;

ENTITY alias_rec;
  object : NAME_TYPE;                              -- type of object
  property : NAME_TYPE;                            -- object property
  system : NAME_TYPE;                              -- system requiring an alias
  alias_list : LIST [0:?] of alias_list_rec;       -- list of aliases
  comment : NAME_TYPE;
END_ENTITY;

END_SCHEMA;
```

6.2.1.5 Annotation Data Schema

This is the EXPRESS model for annotation data. Currently, annotation is limited to text.

EXPRESS Specification :
*)

```
SCHEMA annotation_schema;

REFERENCE FROM rpdtypes_schema;

ENTITY annotation_rec;
  text : STRING;                       -- label
  text_height : DIMENSION;             -- text size
  text_width : DIMENSION;              -- text size
  line_width : DIMENSION;              -- width of text line
  layer : NAME_TYPE;                   -- text layer
  location : POINT_REC;                -- text location
  rotation : INTEGER;                  -- text rotation
  justification : NAME_TYPE;           -- text justification
END_ENTITY;

END_SCHEMA;
```

6.2.1.6 CARI Data Schema

This Express model is in place for Raytheon legacy data for its proprietary Computer Aided Routing of Interconnect (CARI) system. As a generic model this should be eliminated.

EXPRESS Specification :
*)

```
SCHEMA cari_schema;

REFERENCE FROM rpdtypes_schema;

ENTITY cari_rule_rec;
  cari_id : NAME_TYPE;            -- keyword for CARI record
  record : NAME_TYPE;             -- CARI record card image
  comment : NAME_TYPE;            -- pointer to comment string
END_ENTITY;
```

```
END_SCHEMA;
```

6.2.1.7 Class Data Schema

This EXPRESS model defines data entities for classifying signal nets into groups for particular design rules.

EXPRESS Specification :
*)

```
SCHEMA class_schema;

REFERENCE FROM rpdtypes_schema;

ENTITY class_rec;
  group_name : NAME_TYPE;                     -- class identifier
  design_rules : NAME_TYPE;                   -- design rules block
  signal_list : LIST [0:?] of NAME_TYPE;      -- signals in the class
  attribute : LIST [0:?] of ATTRIBUTE_REC;    -- user defined attribute
  comments : LIST [0:?] of STRING;            -- text description
END_ENTITY;

END_SCHEMA;
```

6.2.1.8 Comment Data Schema

This schema defines a *single entity for a comment* a list of comments is kept with each PWB design.

EXPRESS Specification :
*)

```
SCHEMA comment_schema;

REFERENCE FROM rpdtypes_schema;

ENTITY comment_rec;
  comment : NAME_TYPE;
END_ENTITY;

END_SCHEMA;
```

6.2.1.9 Design Rule Data Schema

This EXPRESS schema defines entities for design rules. Design rules are stored in named blocks. Each block except for the GLOBAL block has a Parent name which it inherits from.

EXPRESS Specification :
*)

```
SCHEMA dr_block_schema;

REFERENCE FROM rpdtypes_schema;
```

```
ENTITY substrate_block_rec;
  name : NAME_TYPE;                           -- substrate name
  technology : NAME_TYPE;                     -- technology code
  mode : INTEGER;                             -- code for mode
  layers : INTEGER;                           -- number of layers
  pad_stack_file : NAME_TYPE;                 -- RLD file containing pad
stackups
  layer_model : LIST [0:?] of LAYER_TYPE;     -- layer model names
  separation : LIST [0:?] of INTEGER;         -- spacing between layers
  prepreg_mat : NAME_TYPE;                    -- prepreg material
  substrate_mat : NAME_TYPE;                  -- substrate material
  solder_mat : NAME_TYPE;                     -- solder_mask material
  attribute : LIST [0:?] of ATTRIBUTE_REC;    -- user defined attributes
END_ENTITY;

ENTITY via_spec_rec;
  via_shape : STRING;                         -- default via shape
  via_length : DIMENSION;                     -- default via length
  via_height : DIMENSION;                     -- default via height
END_ENTITY;

ENTITY via_step_rec;
  via_spacing : DIMENSION;                    -- minimum via separation
  via_depth : INTEGER;                        -- maximum via depth
  first_layer : INTEGER;                      -- first stepping layer
  pattern : NAME_TYPE;                        -- stepping pattern
  direction : REAL;                           -- direction for first step
END_ENTITY;

ENTITY min_space_rec;
  line_to_line : INTEGER;                     -- line-to-line spacing
  line_to_pad : INTEGER;                      -- line-to-pad spacing
  pad_to_pad : INTEGER;                       -- pad-to-pad spacing
  line_to_profile : INTEGER;                  -- line-to-profile spacing
  pad_to_profile : INTEGER;                   -- pad-to-profile spacing
END_ENTITY;

ENTITY design_block_rec;
  boundary : LIST [0:?] of vertex_rec;        -- design rules boundary
  layer_t : LAYER_TYPE;                       -- design rules layer
  layer_polarity : NAME_TYPE;                 -- layer polarity codes
  x_grid : LIST [0:?] of REAL;                -- board routing x grid size
  y_grid : LIST [0:?] of REAL;                -- board routing y grid size
  grid_offset : POINT_REC;                    -- routing grid offset
  x_via_grid : LIST [0:?] of REAL;            -- board via x grid size
  y_via_grid : LIST [0:?] of REAL;            -- board via y grid size
  via_grid_offset : POINT_REC;                -- via grid offset
  spacing : min_space_rec;                    -- feature spacing rules
  via_spec : via_spec_rec;                    -- pointer to default via
  via_stepping : via_step_rec;                -- via stepping data
  acid_trap : INTEGER;                        -- acid trap angle
  attribute : LIST [0:?] of ATTRIBUTE_REC;    -- user defined attributes
END_ENTITY;

ENTITY miter_rec;
  angle : DIMENSION;                          -- mitering angle
  length : I_RANGE_REC;                       -- length of miter
END_ENTITY;

ENTITY termination_rec;
  term_type : TOKEN;                          -- type of termination (INPUT |
OUTPUT| DUAL)
  value : REAL;                               -- resistor value in ohms
  unterm : DIMENSION;                        -- max unterminated length
END_ENTITY;

ENTITY necking_rec;
  line_width : DIMENSION;                    -- minimum necked width
  length : I_RANGE_REC;                      -- length of neck
  spacing : DIMENSION;                       -- unnecked spacing between 2
necks
END_ENTITY;

ENTITY parallelism_rec;
  parallel_type : NAME_TYPE;                 -- total or individual
  plane : NAME_TYPE;                         -- coplanar or biplanar
  separation : DIMENSION;                    -- separation threshold between
traces
  limit : DIMENSION;                         -- parallel traces length
threshold
END_ENTITY;

ENTITY shield_rec;
  shield_type : NAME_TYPE;                   -- shielding type: microstrip,
stripline,
                                             -- grounded, guarded, shielded
  signal : NAME_TYPE;                        -- signal shield connected
  cover_width : DIMENSION;                   -- cover width for shield
  strip_width : DIMENSION;                   -- stripline width
  isolation : DIMENSION;                     -- isolation dist
  post_spacing : DIMENSION;                  -- via post space distance
  post_stackup: NAME_TYPE;                   -- stackup for vias for posts
END_ENTITY;

ENTITY signal_block_rec;
  layers : bitmask;                          -- eligible routing layers
  layer_t : LIST [0:?] of LAYER_TYPE;        -- list of layer types
  signal_type : NAME_TYPE;                   -- signal type: power, ground,
ecl, etc.
  line_width : DIMENSION;                    -- default wire line width
  line_shape : NAME_TYPE;                    -- line aperture_shape
  max_length : DIMENSION;                    -- max signal conductor length
  min_length : DIMENSION;                    -- min signal conductor length
  stub : DIMENSION;                          -- max stub length
  net_order : NAME_TYPE;                     -- stringing algorithm: MST,
DAISY, STAR, WIREWRAP
  route_bias : REAL;                         -- routing priority
  clearance : DIMENSION;                     -- net isolation distance
  place_bias : REAL;                         -- placement priority
  via_type : NAME_TYPE;                      -- pad stack for via
  transmission : DIMENSION;                  -- max transmission length
  span : DIMENSION;                          -- driver span
  via_count : INTEGER;                       -- maximum # of vias
  tolerance : DIMENSION;                     -- matched length tolerance
  miter : miter_rec;                         -- corner mitering rules
  termination : termination_rec;             -- terminatin rules
  necking : necking_rec;                     -- necking rules
  parallelism : LIST [0:?] of parallelism_rec; -- parallelism rules
  delay_rule : r_span_rec;                   -- propagation delay rules
  shield_data : shield_rec;                  -- shielding rules
  attribute : LIST [0:?] of ATTRIBUTE_REC;   -- user defined attributes
END_ENTITY;

ENTITY layer_block_rec;
  layer_t : LAYER_TYPE;                      -- design rules layer
  cu_weight : REAL;                          -- copper weight
  thickness : REAL;                          -- thickness of metal
```

```
    impedance : INTEGER;                        -- layer impedence
    purpose : NAME_TYPE;                        -- user define purpose
    attribute : LIST [0:?] of ATTRIBUTE_REC;    -- user defined attributes
  END_ENTITY;

  ENTITY device_block_rec;
    x_grid : LIST [0:?] of REAL;                -- placement grid size
    y_grid : LIST [0:?] of REAL;                -- placement grid size
    grid_offset : POINT_REC;                    -- placement grid offset
    layer_name : LAYER_TYPE;                    -- component placement layer
    via_flag : BOOLEAN;                         -- via inhibit flag
    location_set : NAME_TYPE;                   -- placement location set
    auto_insert : NAME_TYPE;                    -- auto insertion code
    technology : NAME_TYPE;                     -- device technology
    device_bias : REAL;                         -- device affinitity
    thermal_bias : REAL;                        -- thermal affinitity
    space_rule : LIST [0:?] OF NAME_TYPE;       -- placement spaceing rule
    decoupling : DIMENSION;                     -- decoupling distance
    overlap : LIST [0:?] OF NAME_TYPE;          -- placement overlap rule
    wire_bond : I_RANGE_REC;                    -- wire bonding device rules
    aspect : R_RANGE_REC;                       -- aspect ratio for resist
    heat_sink : NAME_TYPE;                      -- heat sink id
    attribute : LIST [0:?] of ATTRIBUTE_REC;    -- user defined attributes
  END_ENTITY;

  ENTITY metal_area_block_rec;
    pin_clearance : DIMENSION;                  -- metal to pin clearance
    via_clearance : DIMENSION;                  -- metal to via clearance
    wire_clearance : DIMENSION;                 -- metal to wire clearance
    conn_number : INTEGER;                      -- connections to each pin
    conn_width : DIMENSION;                     -- width of pin connections
    cutout_flag : BOOLEAN;                      -- flag to generate cutouts
    suppress_flag : BOOLEAN;                    -- unused pad suppression
    show_connect : BOOLEAN;                     -- show pad connections
    default_drill : DIMENSION;                  -- default drill size
    attribute : LIST [0:?] of ATTRIBUTE_REC;    -- user defined attributes
  END_ENTITY;

  ENTITY dr_block_rec;
    block_name : NAME_TYPE;                     -- name of design rule block
    parent_name : NAME_TYPE;                    -- name of parent design rule
block
    substrate_block : substrate_block_rec;      -- substrate rules
    design_block : design_block_rec;            -- design rules
    signal_block : signal_block_rec;            -- signal rules
    layer_block : layer_block_rec;              -- level rules
    device_block : device_block_rec;            -- signal rules
    metal_area_block : metal_area_block_rec;    -- metal area rules
  END_ENTITY;

  END_SCHEMA;
```

6.2.1.10 Gate Data Schema

This schema defines entities for device gates.

<u>EXPRESS Specification :</u>

```
*)

SCHEMA gate_schema;

REFERENCE FROM rpdtypes_schema;
```

```
ENTITY gate_package_rec;
  component : NAME_TYPE;                        -- symbolic component name
  gate_no   : NAME_TYPE;                        -- element number
END_ENTITY;

ENTITY sheet_rec;
  num : NAME_TYPE;                              -- sheet number
  x_location : REAL;                            -- location on sheet
  y_location : REAL;                            -- location on sheet
END_ENTITY;

ENTITY gate_net_rec;
  logic_pin : NAME_TYPE;                        -- logical pin name
  signal : NAME_TYPE;                           -- default net name
END_ENTITY;

ENTITY gate_rec;
  instance : NAME_TYPE;                         -- gate name (handle)
  package : gate_package_rec;                   -- package reference
  old_package : gate_package_rec;               -- original package ref
  gate_swap_code : NAME_TYPE;                   -- swap group name
  swap_inhibit : INTEGER;                       -- gate/pin swapability
  gate_count : INTEGER;                         -- identical gate/device
  sheet : sheet_rec;                            -- schematic location
  comment : NAME_TYPE;                          -- pointer to comment string
  signal_map : LIST [0:?] of gate_net_rec;      -- list of pins and nets
  old_signal_map : LIST [0:?] of gate_net_rec;  -- list of pins and nets
  attribute : LIST [0:?] of attribute_rec;      -- user defined attribute
END_ENTITY;

END_SCHEMA;
```

6.2.1.11 Net Data Schema

This schema defines entities for net signals.

EXPRESS Specification :

```
*)

SCHEMA net_schema;

REFERENCE FROM rpdtypes_schema;
REFERENCE FROM pin_schema;
REFERENCE FROM via_schema;
REFERENCE FROM route_schema;
REFERENCE FROM metal_area_schema;
REFERENCE FROM dr_block_schema;

ENTITY ww_pin_data_rec;
  method : NAME_TYPE;           -- installation method
  code : NAME_TYPE;             -- wire type code
  sequence : INTEGER;           -- wrap sequence
  group : NAME_TYPE;            -- wire group
  length : DIMENSION;           -- xs wire length
  findno : NAME_TYPE;           --
  inst_path : STRING;           -- installation path
END_ENTITY;

ENTITY ww_data_rec;
  run_number : INTEGER;         -- wire wrap run number
  func : NAME_TYPE;             -- net function
```

```
END_ENTITY;

ENTITY ww_pin_pair_rec;
  method : NAME_TYPE;              -- installation method
  code : NAME_TYPE;                -- wire type code
  sequence : INTEGER;              -- wrap sequence
  group : NAME_TYPE;               -- wire group
  length : INTEGER;                -- xs wire length
  findno : NAME_TYPE;
  inst_path : NAME_TYPE;           -- installation path
END_ENTITY;

ENTITY pin_pair_rec;
  t_pin_name : pin_name_rec;     -- to pin name
  f_pin_name : pin_name_rec;     -- from pin name
  t_pin : pin_rec;               -- to pin object
  f_pin : pin_rec;               -- from pin object
  pp_index : INTEGER;            -- index to route object
  pp : route_rec;                -- pointer to route object
  ww_pins : ww_pin_pair_rec;     -- wire wrap pin pair data
END_ENTITY;

ENTITY net_rec;
  name : NAME_TYPE;                                 -- name of net
  design_rules : NAME_TYPE;                         -- design rules block
  signal_type : NAME_TYPE;                          -- signal type
  pin_pairs : LIST [0:?] OF pin_pair_rec;           -- list of pin pairs
  ww_data : ww_data_rec;                            -- wire wrap data
  layer : BITMASK;                                  -- eligible routing layers
  layer_t : LIST [0:?] OF NAME_TYPE;                -- list of layer types
  line_width : DIMENSION;                           -- line width for routing
  line_shape : NAME_TYPE;                           -- line aperture_shape
  max_length : DIMENSION;                           -- minimum total wire
length
  min_length : DIMENSION;                           -- maximum total wire
length
  stub : DIMENSION;                                 -- maximum stub length
  net_order : NAME_TYPE;                            -- stringing algorithm
  clearance : DIMENSION;                            -- net isolation distance
  route_bias : REAL;                                -- routing priority
  place_bias : REAL;                                -- placement priority
  via_type : NAME_TYPE;                             -- absolute pin(via) type
  transmission : DIMENSION;                         -- transmission length
  span : DIMENSION;                                 -- driver span
  via_count : INTEGER;                              -- maximum # of vias
  miter : miter_rec;                                -- corner mitering rules
  termination : termination_rec;                    -- terminatin rules
  necking : necking_rec;                            -- necking rules
  parallelism : LIST [0:?] of parallelism_rec;      -- parallelism rules
  shield : shield_rec;                              -- shielding rules
  pin_names : LIST [0:?] of pin_name_rec;           -- pin names in the net
  pins : LIST [0:?] OF pin_rec;                     -- pin records in the net
  routes : LIST [0:?] of route_rec;                 -- list of net routes
  vias : LIST [0:?] of via_rec;                     -- list of net vias
  metal_areas : LIST [0:?] of metal_area_rec;       -- list of net metal areas
  delay_rule : r_span_rec;                          -- propagation delay rules
  comment : NAME_TYPE;                              -- comment string
  attribute : LIST [0:?] OF ATTRIBUTE_REC;          -- user defined attribute
END_ENTITY;

END_SCHEMA;
```

6.2.1.12 Metal Area Data Schema

This schema defines entities for metal areas (areas of a PWB flooded or meshed with conductor material).

EXPRESS Specification :

```
*)

SCHEMA metal_area_schema;

REFERENCE FROM rpdtypes_schema;
REFERENCE FROM dr_block_schema;

ENTITY cutout_rec;
  cutout_type : NAME_TYPE;                     -- type of cutout
  points : LIST [0:?] of POINT_REC;            -- cutout description
END_ENTITY;

ENTITY metal_area_rec;
  signal : NAME_TYPE;
  metal_area_type : NAME_TYPE;                 -- type of metal area
  style : NAME_TYPE;                           -- style of metal area
  design_rules : dr_block_rec;                 -- name of design rule block
  aperture : DIMENSION;                        -- apperature for photoplot
  spacing : DIMENSION;                         -- line spacing in photoplot
  layer : INTEGER;                             -- layer for metal area
  cutout_shape : NAME_TYPE;                    -- shape for pin cutouts
  origin : POINT_REC;                          -- boundary origin
  boundary : LIST [0:?] of POINT_REC;          -- boundary description
  user_cutouts : LIST [0:?] of cutout_rec;     -- defined cutouts
  auto_cutouts : LIST [0:?] of cutout_rec;     -- generated cutouts
  comment : NAME_TYPE;                         -- comment string
  attribute : LIST [0:?] of ATTRIBUTE_REC;     -- user defined attribute
END_ENTITY;

END_SCHEMA;
```

6.2.1.13 Part Data Schema

This schema defines the electrical characteristics of the PWB components.

EXPRESS Specification :

```
*)

SCHEMA part_schema;

REFERENCE FROM rpdtypes_schema;

ENTITY pin_map_rec;
  logic_pin : NAME_TYPE;                   -- logical pin name
  component_pin : NAME_TYPE;               -- component pin name
  pin_swap_code : NAME_TYPE;               -- pin swap group
END_ENTITY;

ENTITY element_rec;
  elem_no : NAME_TYPE;                     -- element number
  elem_swap : NAME_TYPE;                   -- element Swap Code
  pin_map : LIST [0:?] OF pin_map_rec;     -- element to device pin map
END_ENTITY;
```

```
ENTITY geo_data_rec;
  rev : NAME_TYPE;                         -- pin data rev
  modn : NAME_TYPE;                        -- pin data mod
  clear_z : DIMENSION;                     -- component CLEARZ
  height : DIMENSION;                      -- component HEIGHT
  length : DIMENSION;                      -- component LENGTH
  width : DIMENSION;                       -- clib component WIDTH
  hsx : DIMENSION;                         -- clib HSX pin spacing
  hsy : DIMENSION;                         -- clib HSY pin spacing
  mass : REAL;                             -- component MASS
  pin_offset : point_rec;                  -- pin offset
END_ENTITY;

ENTITY op_data_rec;
  rev : NAME_TYPE;                         -- pin data rev
  modn : NAME_TYPE;                        -- pin data mod
  power_dissip : REAL;                     -- power dissipation
  max_power_dissip : REAL;                 -- max power dissipation
  peak_power : REAL;                       -- peak power
  min_power : REAL;                        -- min power
END_ENTITY;

ENTITY therm_data_rec;
  rev : NAME_TYPE;                         -- pin data rev
  modn : NAME_TYPE;                        -- pin data mod
  emit : REAL;
  rsbtm : REAL;
  rsjb : REAL;
  rsjc : REAL;
  rstop : REAL;
  spht : REAL;
  jtm : REAL;
  thermal_type_code : INTEGER;
  thermal_type : NAME_TYPE;
END_ENTITY;

ENTITY pin_time_rec;
  min : REAL;
  typical : REAL;
  max : REAL;
END_ENTITY;

ENTITY input_current_rec;
  iil : REAL;                             -- low current
  iih : REAL;                             -- high current
END_ENTITY;

ENTITY input_voltage_rec;
  vil : REAL;                             -- low voltage
  vih : REAL;                             -- high voltage
END_ENTITY;

ENTITY output_current_rec;
  iol : REAL;
  ioh : REAL;
  iozl : REAL;
  iozh : REAL;
END_ENTITY;

ENTITY output_voltage_rec;
  vol : REAL;                             -- low voltage
  voh : REAL;                             -- high voltage
  vol_min : REAL;                         -- min voltage
  voh_max : REAL;                         -- max voltage
END_ENTITY;

ENTITY bi_pin_rec;
  input_current : input_current_rec;
  input_voltage : input_voltage_rec;
  output_current : output_current_rec;
  output_voltage : output_voltage_rec;
END_ENTITY;

ENTITY in_pin_rec;
  input_current : input_current_rec;
  input_voltage : input_voltage_rec;
END_ENTITY;

ENTITY ou_pin_rec;
  ou_config_code : INTEGER;
  ou_config : NAME_TYPE;
  output_current : output_current_rec;
  output_voltage : output_voltage_rec;
END_ENTITY;

ENTITY pin_data_rec;
  rev : NAME_TYPE;                         -- pin data rev
  modn : NAME_TYPE;                        -- pin data mod
  pin_number : NAME_TYPE;                  -- component pin number
  pin_name : NAME_TYPE;                    -- component pin name
  pin_swap_code : NAME_TYPE;               -- pin swap group name
  pin_offset : POINT_REC;                  -- center of the pin relative to
the origin of the device
  capacitance : REAL;
  fall_time : pin_time_rec;                -- rise time
  rise_time : pin_time_rec;                -- fall time
  pin_type : NAME_TYPE;                    -- B, I, O
  bi_pin : bi_pin_rec;                     -- bi_directional pin data
  in_pin : in_pin_rec;                     -- input pin data
  ou_pin : ou_pin_rec;                     -- output pin data
END_ENTITY;

ENTITY prop_delay_rec;
  rev : NAME_TYPE;                         -- pin data rev
  modn : NAME_TYPE;                        -- pin data mod
  pin_name_start : NAME_TYPE;
  pin_name_end : NAME_TYPE;
  pin_num_start : NAME_TYPE;
  pin_num_end : NAME_TYPE;
  phl : REAL;
  plh : REAL;
  unateness : NAME_TYPE;
END_ENTITY;

ENTITY part_rec;
  part : NAME_TYPE;                            -- part name
  technology : NAME_TYPE;                      -- device technology
  spice_model : NAME_TYPE;                     -- spice model for the device
  heat_flag : BOOLEAN;                         -- heat sensitivity flag
  stat_flag : BOOLEAN;                         -- static sensitivity flag
  polar_flag : BOOLEAN;                        -- polar component flag
  part_type : NAME_TYPE;                       -- component type
  part_class : NAME_TYPE;                      -- component class
  description : STRING;                        -- component description
  mil_spec : NAME_TYPE;                        -- component mil_spec name
  findno : NAME_TYPE;                          -- component find number
  tolerance : NAME_TYPE;                       -- component tolerance
  value : NAME_TYPE;                                -- component value
  mech_name : NAME_TYPE;                            -- mechanical name
  manufacturer : NAME_TYPE;                         -- part manufacturer
  elements : LIST [0:?] OF element_rec;             -- list of elements in part
  geo_data : geo_data_rec;                          -- geometry data
  op_data : op_data_rec;
  therm_data : therm_data_rec;                      -- thermal data
  pin_data : LIST [0:?] OF pin_data_rec;            -- pin data
  delay_data : LIST [0:?] OF prop_delay_rec;        -- delay data
  comment : NAME_TYPE;                              -- comment string
  attribute : LIST [0:?] OF ATTRIBUTE_REC;          -- user defined attributes
END_ENTITY;

END_SCHEMA;
```

6.2.1.14 Pin Data Schema

This schema defines entities for component pins instantiated on the PWB.

EXPRESS Specification :
*)

```
SCHEMA pin_schema;

REFERENCE FROM rpdtypes_schema;

TYPE function_type = STRING(1) FIXED; END_TYPE;
     -- I for input or source
     -- O output or sink
     -- B bidirectional
     -- T pin on a terminating resistor

ENTITY load_data_rec;
  power : LOADING_TYPE;                          -- power loading data
  voltage : LOADING_TYPE;                        -- voltage loading data
  current : LOADING_TYPE;                        -- current loading data
  temperature : LOADING_TYPE;                    -- temperature loading data
END_ENTITY;

ENTITY pin_rec;
  pin : NAME_TYPE;                               -- pin name
  signal : NAME_TYPE;                            -- signal name
  offset : POINT_REC;                            -- pin offset from origin
  location : POINT_REC;                          -- pin location on board
  rotation : REAL;                               -- pin rotation in degrees
  range : BITMASK;                               -- pin depth
  suppression : BITMASK;                         -- pad suppression mask
  func : FUNCTION_TYPE;                          -- pin function code
  stepping : REAL;                               -- first stepping direction
  pin_type : NAME_TYPE;                          -- absolute pin type
  swap_inhibit : INTEGER;                        -- gate/pin swapability
  load_data : load_data_rec;                     -- pin loading data
  comment : NAME_TYPE;                           -- comment string
  attribute : LIST [0:?] of ATTRIBUTE_REC;-- user defined attributes
END_ENTITY;

END_SCHEMA;
```

6.2.1.15 Conductor Routing Data Schema

This schema defines entities for conductor routes of net signals.

EXPRESS Specification :

```
*)

SCHEMA route_schema;

REFERENCE FROM rpdtypes_schema;
REFERENCE FROM net_schema;
REFERENCE FROM pin_schema;

ENTITY segment_rec;
  x : DIMENSION;                        -- x coord of point on the path
  y : DIMENSION;                        -- y coord of point on the path
  radius : INTEGER;                     -- for circular segment
  segment_width : DIMENSION;            -- the width of the segment
END_ENTITY;

ENTITY ww_route_data_rec;
  revision : NAME_TYPE;                 -- wire revision
  sequence : INTEGER;                   -- wire wrap sequence
  bends : LIST [0:?] of POINT_REC;      -- wire wrap bend points
END_ENTITY;

ENTITY route_rec;
  signal : NAME_TYPE;                   -- associated signal name
  route_type : NAME_TYPE;               -- type of connection
  status : NAME_TYPE;                   -- path status
  target_name : pin_name_rec;           -- assigned target pin name
  object_name : pin_name_rec;           -- assigned object pin name
  target_pin : pin_rec;                 -- assigned target pin
  object_pin : pin_rec;                 -- assigned object pin
  target_loc : POINT_REC;               -- coordinates of the target
  object_loc : POINT_REC;               -- coordinates of the object
  protect : BOOLEAN;                    -- path protection flag
  target_layer : INTEGER;               -- assigned starting layer
  object_layer : INTEGER;               -- assigned ending layer
  path : LIST [0:?] OF segment_rec;     -- list of path segments
  shield_id : INTEGER;                  -- code for linking shielding
  pin_pair_index : INTEGER;             -- link to pin-pair data
  pin_pair : pin_pair_rec;              -- link to pin-pair data
  ww_data : ww_route_data_rec;          -- wire wrapping data
  comment : NAME_TYPE;
END_ENTITY;

END_SCHEMA;
```

6.2.1.16 Via Data Schema

This schema defines entities for signal net vias.

EXPRESS Specification :

```
*)

SCHEMA via_schema;

REFERENCE FROM rpdtypes_schema;
REFERENCE FROM dr_block_schema;
REFERENCE FROM net_schema;

ENTITY via_rec;
  signal : NAME_TYPE;                          -- name of signal net
  location : POINT_REC;                        -- board coordinates
  rotation : REAL;                             -- via rotation in degrees
  range : BITMASK;                             -- pin depth
  suppression : BITMASK;                       -- pad suppression mask
  via_type : NAME_TYPE;                        -- absolute via type
  via_use : NAME_TYPE;                         -- special via use
  shield_id : INTEGER;                         -- code for linking shielding
  shield : shield_rec;                         --
  comment : NAME_TYPE;                         -- comment string
  attribute : LIST [0:?] of ATTRIBUTE_REC;     -- user defined attributes
END_ENTITY;

END_SCHEMA;
```

6.2.1.17 Library Cross Reference Data Schema

This schema defines entities for the device cross references.

EXPRESS Specification :

```
*)

SCHEMA xref_schema;

REFERENCE FROM rpdtypes_schema;
REFERENCE FROM pin_schema;

ENTITY xref_rec;
  symbolic : NAME_TYPE;                       -- symbolic name
  old_symbolic : NAME_TYPE;                   -- old symbolic name
  model : NAME_TYPE;                          -- mechanical model name
  location : POINT_REC;                       -- board location
  mirror : INTEGER;                           -- mirror flag
  rotation : REAL;                            -- rotation flag
  symbolic_flag : BOOLEAN;                    -- symbolic pin names used flag
  external : BOOLEAN;                         -- connector flag
  usa_device : NAME_TYPE;                     -- USA device names
  physical : NAME_TYPE;                       -- CLIB device name
  raytheon : NAME_TYPE;                       -- raytheon part number
  design_rules : NAME_TYPE;                   -- design rules block
  layer : NAME_TYPE;                          -- component placement layer
  via_flag : BOOLEAN;                         -- inhibit via under device
  location_set : NAME_TYPE;                   -- placement location set
  auto_insert : NAME_TYPE;                    -- auto insertion code
  swap_inhibit : INTEGER;                     -- gate/pin swapability code
  fix : BOOLEAN;                              -- fixed placement flag
  device_bias : REAL;                         -- device affinitity
  thermal_bias : REAL;                        -- thermal affinity
  coupling : LIST [0:?] of NAME_TYPE;         -- placement coupled devices
  decoupling : INTEGER;                       -- decoupling distance
  space_rule : LIST [0:?] of NAME_TYPE;       -- placement spaceing rule
  overlap : LIST [0:?] of NAME_TYPE;          -- placement overlap rule
  heat_sink : NAME_TYPE;                      -- heat sink name
  load_data : load_data_rec;                  -- loading data
  comment : NAME_TYPE;                        -- comment string
  attribute : LIST [0:?] of attribute_rec;    -- user defined attributes
END_ENTITY;

END_SCHEMA;
```

6.2.2 PWB Design Data EXPRESS-G Model

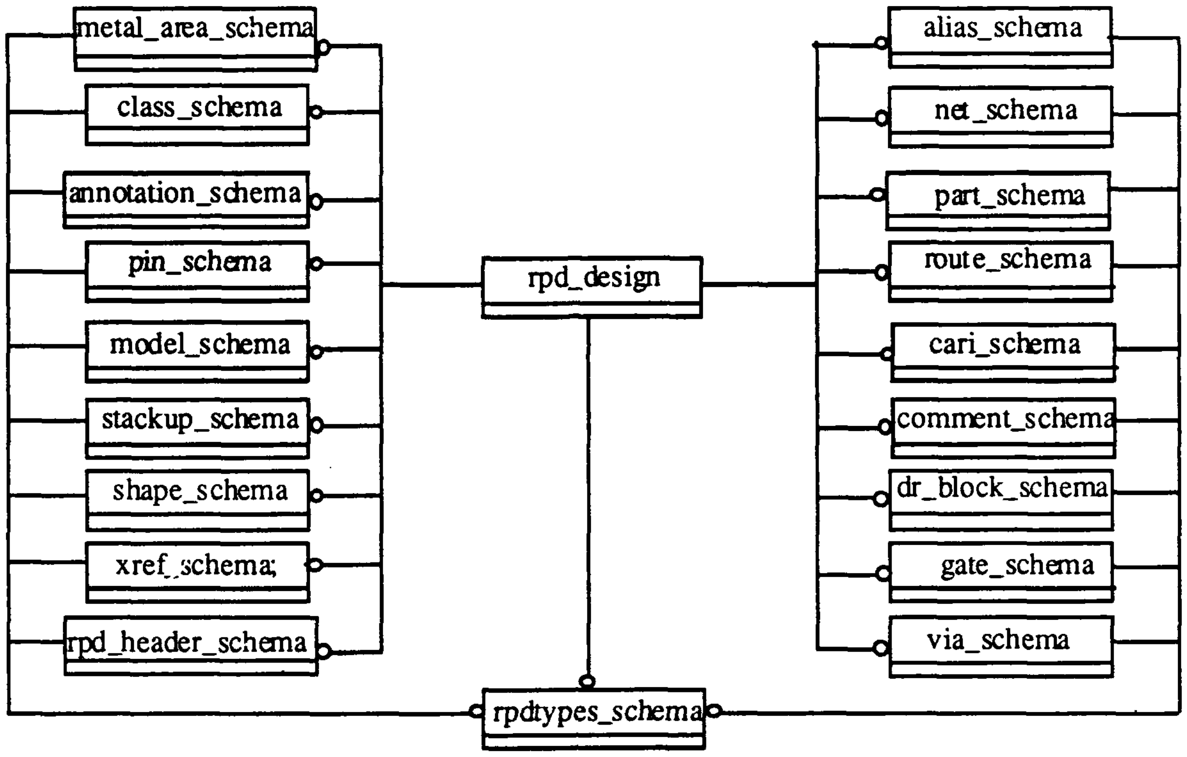


Figure 6.2-1 PWB Schema Level EXPRESS-G Model

6.2.3 Electronic Component Library Data Model

6.2.3.1 Component Model Data Schema

This schema defines entities for modeling PWB components.

<u>EXPRESS Specification :</u>

```
*)

SCHEMA model_schema;

REFERENCE FROM rpdtypes_schema;
REFERENCE FROM rpd_header_schema;
REFERENCE FROM stackup_schema;

ENTITY rev_data_rec;
  issue_date : NAME_TYPE;                   -- date of issue
  revision : NAME_TYPE;                     -- revision number
  eco : NAME_TYPE;                          -- latest eco number
  eco_date : NAME_TYPE;                     -- date of latest eco
END_ENTITY;

ENTITY dev_origin_rec;
  origin_type : NAME_TYPE;                  -- origin types
  center : POINT_REC;                       -- device center
  offset : POINT_REC;                       -- placement offset
  mirror : INTEGER;                        -- reflection code
END_ENTITY;

ENTITY label_rec;
  text : STRING;                           -- label text
  height : DIMENSION;                      -- text size
  width : DIMENSION;                       -- text size
  location : POINT_REC;                    -- text location
  rotation : INTEGER;                      -- text rotation
  line_width : DIMENSION;                  -- width of text line
  justify : NAME_TYPE;                     -- text justification
END_ENTITY;

ENTITY boundary_rec;
  boundary_type : NAME_TYPE;               -- type of boundary
  shape : NAME_TYPE;                       -- boundary outline shape
  outline : LIST [0:?] of VERTEX_REC;      -- boundary outline vertices
  layers : LIST [0:?] of NAME_TYPE;        -- boundary layers
END_ENTITY;

ENTITY obstruction_rec;
  obstruction_type : NAME_TYPE;            -- type of obstruction
  shape : SHAPE_TYPE;                      -- outline shape
  outline : LIST [0:?] of VERTEX_REC;      -- pad outline
  layers : LIST [0:?] of LAYER_TYPE;       -- pad layers
  blocking : LIST [0:?] of BLOCKING_TYPE;  -- blocking codes
END_ENTITY;

ENTITY device_rec;
  symbolic : NAME_TYPE;                  -- symbolic name
  physical : NAME_TYPE;                  -- physical name
  model : NAME_TYPE;                     -- mechanical model name
  location : POINT_REC;                  -- location on board
  rotation : REAL;                       -- rotation in degrees
  mirror : INTEGER;                      -- mirror flag
END_ENTITY;

ENTITY dev_pin_rec;
  physical : STRING;                     -- physical pin name (must be
string of integers)
  symbolic : NAME_TYPE;                  -- symbolic pin name
  location : POINT_REC;                  -- pin location
  drill : DIMENSION;                     -- default drill size
  stackup_name : NAME_TYPE;              -- pad stackup name
  stackup : STACKUP_REC;                 -- pad stackup record
  rotation : REAL;                       -- stackup rotation
  offset : POINT_REC;                    -- stackup offset
  stepping : INTEGER;                    -- first stepping direction
END_ENTITY;

ENTITY thermal_rec;
  thermal_type : NAME_TYPE;              -- type of thermal relief
  width : DIMENSION;                     -- line width
  spacing : DIMENSION;                   -- line spacing
  stackup_name : NAME_TYPE;              -- stackup name
  stackup : STACKUP_REC;                 -- stackup record
END_ENTITY;

ENTITY package_rec;
  package_type : NAME_TYPE;              -- package type
  category : NAME_TYPE;                  -- package category
  orientation : NAME_TYPE;               -- package orientation
  distance : DIMENSION;                  -- pin row separation
  depth : DIMENSION;                     -- package depth
  height : DIMENSION;                    -- package height
  width : DIMENSION;                     -- package width
  lead : DIMENSION;                      -- package lead diameter
  fix : BOOLEAN;                         -- fixed device flag
  body_diameter : DIMENSION;             -- package body diameter
  span : DIMENSION;                      -- package pin span
  insert : NAME_TYPE;                    -- package insertion code
  mechanical : BOOLEAN;                  -- mechanical device flag
  auto_ww_offset : POINT_REC;            -- automatic wirewrap offset
  auto_ww_trp : INTEGER;                 -- automatic wirewrap initial trp
  semi_ww_offset : POINT_REC;            -- semiautomatic wirewrap offset
  semi_ww_trp : INTEGER;                 -- semiautomatic wirewrap initial
trp
END_ENTITY;

ENTITY model_rec;
  header : header_rec;                           -- pointer to header record
  mm_name : NAME_TYPE;                           -- mechanical model name
  rev_data : rev_data_rec;                       -- revision data
  origin : dev_origin_rec;                       -- origin data
  package : package_rec;                         -- packafing data
  labels : LIST [0:?] of label_rec;              -- list of labels
  boundaries : LIST [0:?] of boundary_rec;       -- list of boundaries
  obstructions : LIST [0:?] of obstruction_rec;  -- list of obstructions
  devices : LIST [0:?] of device_rec;            -- list of devices
  pins : LIST [0:?] of dev_pin_rec;              -- list of pins
  thermals : LIST [0:?] of thermal_rec;          -- list of thermal reliefs
  comments : LIST [0:?] of STRING;               -- list of comments
  attribute : LIST [0:?] of attribute_rec;       -- list of user defined
attributes
END_ENTITY;

END_SCHEMA;
```

6.2.3.2 Pad Stack Data Schema

This schema defines entities for pin and via pad stackups. Various pad shapes for each layer are combined. The layer assignments are then combined to form the padstack.

EXPRESS Specification :

```
*)

SCHEMA stackup_schema;

REFERENCE FROM rpdtypes_schema;
REFERENCE FROM shape_schema;

ENTITY pad_rec;
  pad_name : NAME_TYPE;                  -- shape name
  pad_shape : PAD_SHAPE_REC;             -- pad shapes
  func : NAME_TYPE;                      -- pad function
END_ENTITY;

ENTITY pad_stack_rec;
  model : NAME_TYPE;                     -- layer model
  offset : POINT_REC;                    -- pad offset
  pad_list : LIST [0:?} of pad_rec;      -- pad_names
END_ENTITY;

ENTITY stackup_rec;
  stack_name : NAME_TYPE;                -- name of stackup
  pad_stack : LIST [0:?] of pad_stack_rec; -- pad stackups
  drill : INTEGER;                         -- default drill size
  comments : LIST [0:?] of STRING;         -- list of comments
END_ENTITY;

END_SCHEMA;
```

6.2.3.3 Pad Shape Data Schema

This schema defines entities for pin and via pad shapes.

<u>EXPRESS Specification :</u>

```
*)

SCHEMA shape_schema;

REFERENCE FROM rpdtypes_schema;

ENTITY shape_rec;
  shape : NAME_TYPE;                     -- shape type
  width : DIMENSION;                     -- aperature width
  outline : LIST [0:?] of VERTEX_REC;    -- shape description
END_ENTITY;

ENTITY pad_shape_rec;
  name : NAME_TYPE;                      -- shape name
  pads : LIST [0:?] of shape_rec;        -- pad shapes
END_ENTITY;

END_SCHEMA;
```

6.2.4 Electronic Component Library Data EXPRESS-G Model

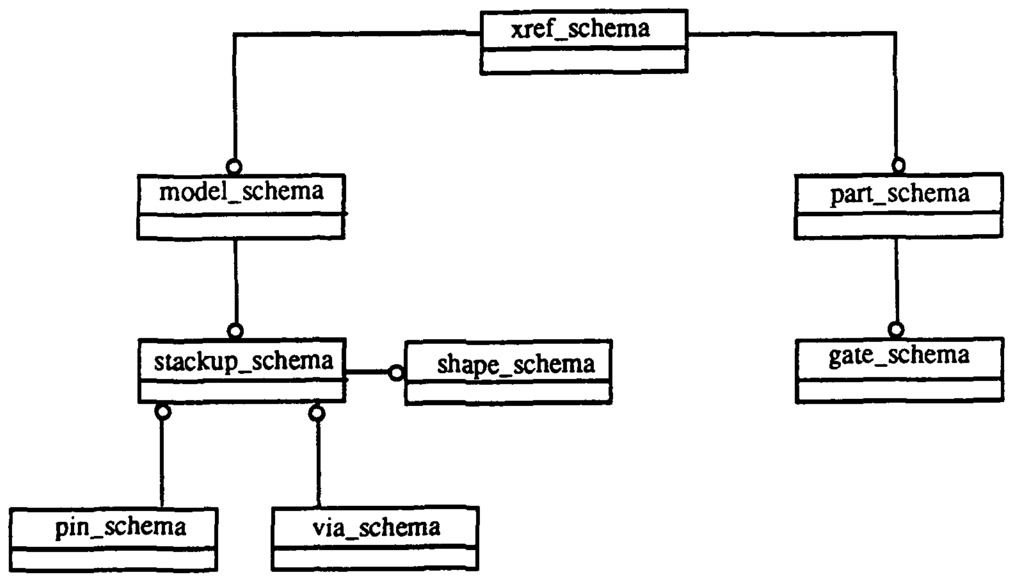


Figure 6.2-2 Component Data EXPRESS-G Schema

7. Requirements Traceability

Provided below is a table which maps the applicable Software Design Specification section numbers to the corresponding Functional Requirements and Measure of Performance document section numbers (refer to reference 4).

Design Spec. Sections	Functional Spec. Sections
3.2.1.1	3.1.1
3.2.1.2, 5.6	3.2.2
3.2.2, 4.1.1, 5.1	3.1.2
3.2.3, 4.1.2, 5.2	4.3
3.2.4.1, 5.3	3.2.1
3.2.4.2, 5.3	3.2.3
3.2.4.3, 5.3	3.2.4
3.2.5, 5.4	3.2.5
4.	3.3
6.	3.5

8. Notes

8.1 Acronyms

CAEO	Computer Aided Engineering Operations
CDRL	Contract Data Requirements List
CERC	Concurrent Engineering Research Center
CM	Communications Manager
DARPA	Defense Advanced Research Projects Agency
DBMS	Database Management System
DFMA	Design for Manufacturing and Assembly
DICE	DARPA Initiative In Concurrent Engineering
ISO	International Standards Organization
MEL	Mechanical Engineering Laboratory
MO	Manufacturing Optimization
MSD	Missile Systems Division
MSL	Missile Systems Laboratories
OOD	Object Oriented Design
OSF	Open Software Foundation
PCB	Project Coordination Board
PWA	Printed Wiring Assembly
PWB	Printed Wiring Board
PWF	Printed Wiring Fabrication
RAPIDS	Raytheon Automated Placement and Interconnect Design System
RM	Requirements Manager
ROSE	Rensselaer Object System For Engineering
SDAI	STEP Data Access Interface
STEP	Standard for Exchange of Product Model Data

Distribution List

DPRO-Raytheon
C/O Raytheon Company
Spencer Lab., Wayside Ave.
(one copy of each report)

Defense Advanced Research Projects Agency
ATTN: Defense Sciences Office; Dr. H. Lee Buchanan
Virginia Square Plaza
3701 N. Fairfax Drive
Arlington, VA. 22203-1714
(one copy of each report)

Defense Advanced Research Projects Agency
ATTN: Electronic Systems Technology Office; Capt. Nicholas J. Naclerio, USAF
Virginia Square Plaza
3701 N. Fairfax Drive
Arlington, VA. 22203-1714
(one copy of each report)

Defense Advanced Research Projects Agency
ATTN: Contracts Management Office; Mr. Donald C. Sharkus
Virginia Square Plaza
3701 N. Fairfax Drive
Arlington, VA. 22203-1714
(one copy of each report)

Defense Technical Information Center
Building 5, Cameron Station
ATTN: Selections
Alexandria, VA 22304
(two copies of each report)